LIBRARY OF THE
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

510.84
Il6r
no. 511-516
cop. 2

Digitized by the Internet Archive
in 2013

http://archive.org/details/graphstructuretr514schw

I16r
no. 514
cop. 2

UIUCDCS-R-72-514 COO-2118-0031

A GRAPH-STRUCTURE TRANSFORMATION MODEL FOR PICTURE PARSING

By

John C. Schwebel

May 1972

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

THE LIBRARY OF THE
SEP 5 1972
UNIVERSITY OF ILLINOIS
AT URBANA CHAMPAIGN

The person charging this material is responsible for its return to the library from which it was withdrawn on or before the **Latest Date** stamped below.

Theft, mutilation, and underlining of books are reasons for disciplinary action and may result in dismissal from the University.

UNIVERSITY OF ILLINOIS LIBRARY AT URBANA-CHAMPAIGN

NOV 1 1972

JUN 7 RECD

L161—O-1096

COO-2118-0031

UIUCDCS-R-72-514

A GRAPH-STRUCTURE TRANSFORMATION MODEL FOR PICTURE PARSING

By

John C. Schwebel

May 1972

Department of Computer Science
University of Illinois
Urbana, Illinois

This work was supported by Contract AT(11-1)-2118 with the U.S. Atomic Energy Commission.

A GRAPH-STRUCTURE TRANSFORMATION MODEL FOR PICTURE PARSING

BY

JOHN CHARLES SCHWEBEL

B.Math., University of Minnesota, 1964
M.S., University of Illinois, 1966

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1972

Urbana, Illinois

A GRAPH-STRUCTURE TRANSFORMATION MODEL FOR PICTURE PARSING

John Charles Schwebel, Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 1972

This paper develops a graph-structure model for global level picture representation and picture parsing. Parsing procedures are viewed as series of graph transformations which operate on the graph-structure representation.

Graph structures and graph-structure transformations are defined as having properties to facilitate their application to picture parsing. Reversibility and information loss are defined for transformations. Some representative graph transformations are presented after an enumeration of the basic graphs on which they operate.

Simple embeddings (of a transformed graph within the original graph structure) using branches (relations) between a node outside and nodes within the transformed graph are considered. Embedding types are defined as logical functions of branches. Then, composite embeddings between transformed graphs under composite-forming transformations are studied. Composite embedding types are used to infer relations between objects from relations between sub-objects. Implications between embedding types are shown. Picture processing examples are also given.

Next, transformations which consist of operations on structured sets are considered. An application here is the merging or partitioning of objects in picture processing, where composite-forming transformations whose root operations are unions or intersections of set elements are employed. To obtain the necessary structure, the sets are assumed to be lattices. Relations between lattice elements are characterized by the way they act when the lattice is transformed. This is done by defining lattice operational properties of a relation. Relationships between the general embedding types and the lattice operational properties are shown.

The application of transformations for structural inference of simple descriptions is demonstrated.

A Structure Operation Language, SOL, is defined for the computer implementation of the graph structure model. The language is intended to provide a small number of root operations in order to simply express structure transformations on multi-relational graph structures, and to provide a means for experimentation with heuristic scene segmentation strategies.

The generality of the presented model allows the expression of structural inference techniques on multi-graph representations, which will undoubtedly have utility beyond the immediate application to scene analysis.

ACKNOWLEDGEMENT

The author expresses his sincere appreciation to his thesis advisor, Professor Bruce H. McCormick, for his constant aid, encouragement, and his ready availability as a source of dynamic ideas, many of which are in this paper.

Lasting gratitude is also due to my wife, Carolyn, for typing the thesis and especially for needed encouragement during the past year, both of which were necessary for the completion of this thesis. She may now anticipate being repaid in kind as she undertakes a similar folly.

Special thanks go to Mr. Stanley Zundo for a rush job in preparing the drawings.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1 Nature of the Problem

Our long range goal can be viewed as this: to design a system which can infer the syntactic and semantic structure of its visual environment from selected instances of objects and scenes from the environment. Implicit here is the prior existence of parallel processes for scene segmentation.

We identify three subareas of the global recognition problem: Picture Representation, Descriptive Pattern Analysis, and Processing Languages. As asserted above, our exclusive concern will be with global aspects of processing, i.e. some preprocessing is assumed on the picture before it is treated by this model. These processes typically consist of operations which can abstract low-level primitive objects as well as find relations between objects of the scene.

Fundamental to the problem of developing higher level picture processing procedures is the necessity to find a suitable picture representation for algorithms and data. This representation must both represent the hierarchical structures of elements with attributes and interrelate these by binary relations. Basic to our model are a graph-structured data representation and graph-transformational parsing procedures.

Perhaps one of the most important criteria for any

successful picture processing system is flexibility. It is relatively easy to specify a standard grammar to recognize, for example, a "house." But a successful system must be able to recognize "houses" of widely varying types. It is also desirable that the recognition of varying forms of one object be achieved by progressing down a nearly identical path of procedural steps. Flexibility also means the ability to handle objects which may be incompletely specified. The graph-structural representation model has the following features which facilitate flexibility:

Subgraphs may be expressions of topological relations which permit a deformable model.

Attribute values associated with each object may be used to tie down the model to concrete instances.

Interaction or propagation of information between parsing levels, which is needed to identify objects in context, is readily expressible.

Descriptive pattern analysis attempts to build descriptions of patterns. We refer to the procedures which form the core of this analysis as parsing procedures. The central problems attacked by these procedures are the location, isolation, and identification of objects in a picture. These problems have been referred to as the "pattern recognition problem"; however, we believe that they can be better delineated as the problems of segmenting, clustering, and parsing objects in a picture. The current lack of computational sophistication in attacking these problems is

attested to by the seemingly elementary (by human standards) image processing that systems today are able to perform.

In our efforts to develop parsing procedures, we have tried to find techniques applicable to the largest class of pictures first, and specialize to other procedures only when forced to do so. As a result we divide clustering procedures into two categories: first, the lower-level segmentation and clustering procedures for object isolation, and second, higher-level parsing procedures utilizing class-specific rules to identify objects within the global stuctural model.

The first catgory includes classical uni-relational graph-theoretic clustering techniques as well as the newer methods of multi-relational clustering, subgraph replacement, and graph transformational rules.

The second category involves heuristic search and global inference techniques applied to the graph structure. These seek to identify where to dynamically modify the structure and to locate optimum sites for object identification. The goal here is to select first for identification those composite structures most likely to be valid objects.

The abstract labeled and directed graph, whose nodes represent objects and whose edges represent arbitrary binary relations between objects allows recognition procedures based on graph transformations. The recognition process can then be viewed as replacement rules operating on graphs.

An important class of procedures forms composites from objects in a "name-independent" manner, i.e. without having to match each element to an explicitly named prototype. These procedures can choose appropriate graph transformations given relevant properties of the relations between items. This class of procedures may be used in the lower levels of analysis in order to increase the efficiency over a model-driven analyzer.

Set-replacement rules used to build up structure are generalizations of grammatical productions. Predicates may be used to specify when a rule may be applied. This process is called "composite formation." A replacement rule forms a composite element from subelements of the graph. The new graph formed by new composite elements and new relations involving these elements is then the candidate for further composite formation. For successful recognition, this process should tend toward a "recognizable graph," i.e. a graph which has somehow been specified as an acceptable goal.

The general structure transformation schema can now be specified by an ordered production system whose entry points correspond to root graph operations. For example the root operation of creating a branch, or merging two elements,which may be dictated by the current picture segmentation strategy, specifies an entry point into the production schema and thus determines possible graph transformations. These graph transformations can then specify actions calling for other root operations.

The development of processing languages is considered a necessary part of this problem. Currently, adequate languages exist to express algorithms which operate on the array representation of pictures as binary valued elements with neighborhood connectivity relationships. The next and most natural abstraction from the array representation is a graph structure, by means of which many scene segmentation algorithms can be simply expressed. We seek to develop theory for the level of picture processing operations which can be represented as structure transformations. A structure operational language is a helpful tool for precisely specifying and for experimenting with heuristic scene segmentation strategies.

In summary, the problem dealt with here is to develop representations and parsing procedures for global level picture processing. The model to be developed should allow the inference of structural models from examples and counter-examples and permit relational inference to choose candidates for composite formation, and to embed or extend relations to newly formed elements. Successful strategies for structural inference using multi-graph scene representation are undoubtedly techniques which will have utility beyond their immediate application to scene analysis.

1.2 Scope of this Research

In the present work we have first attempted to define

the theoretical framework in which transformations on graphs of related objects are applied. Chapter Two defines graph structures and graph structure transformations, which are general mappings between structures. Necessary properties of the transformations are developed. Reversibilty and information loss are defined for transformations.

A logical study of graph transformations is begun by enumerating basic graphs which may be candidates for a transformation and by defining their representative graph transformations.

Later in Chapter Two, simple types of relation embeddings (which specify how exterior branches are moved under a transformation) between one node outside, to nodes within the graph being transformed are defined. Assuming a simple embedding type, root transformations are used to imply more complex transformations. Implications between transformations are also considered.

Chapter Three restricts attention to composite-forming graph transformations: those which are used to parse a structure. Composite embedding types of relations (between nodes within different graphs being transformed) are defined. Implications between embedding types are shown.

Then transformations which consist of operations on structured sets are considered. An application here is the merging or partitioning of objects in picture processing, where composite-forming transformations whose root operations

are unions or intersections of set elements are employed. To obtain the necessary structure, the sets are assumed to be lattices. Relations between lattice elements are characterized by the way they act when the lattice is transformed. This is done by defining lattice operational properties of a relation. Finally relationships between the general embedding types and the lattice operational properties are considered.

Chapter Four treats questions concerning the application of series of graph transformations which attempt to obtain a simple structured description of an object.

In Chapter Five we have specified a Structure Operation Language, SOL, in order to simply and conveniently express operations on multi-relational graph structures which allow an arbitrary number of variables to be associated with any structure element. The language has a small number of root operations which are sufficient for expressing structure transformations, and has been specified with the goal of embedding it in a higher-level procedural language.

Chapter Six contains conclusions and suggestions for further possibly fruitful research.

1.3 Related Work

1.3.1 Preliminary Remarks

This brief survey is divided to reflect the three main (and unavoidably overlapping) parts of our problem: Picture Representation, Descriptive Pattern Analysis, and Processing Languages.

Thus, a syntactical description may represent a class of pictures, which is analyzed by means of a processing language or by a syntax-directed processor keyed to the description. Section 1.3.2 emphasizes the representation used; section 1.3.3 the method of analysis; and section 1.3.4 available processing languages. Only recognition languages fall in the processing area; descriptive languages have been placed in the representation area.

We are exclusively concerned with global level picture processing, (which encompasses concepts important to both pattern recognition and computer graphics). Low-level representations, modes of analysis and associated languages (such as analyzing an array of picture points by local operation using a plane-parallel processing language) are not included in this survey. For a general survey of picture processing see Rosenfeld (1). Also, for surveys primarily on unstructured feature extraction techniques, see Nagy (2) and Levine (3).

1.3.2 Picture Representation

The representations used to describe and process pictures range from unstructured models and implied relational linguistic models,which implicitly assume relations between objects, to explicit relational graph models.

Unstructured models use some type of attribute, object, value triplets to describe objects, as in Paul (4). This is a useful concept, used also in nonpictorial data-base models as in Hsiao and Harary (5), but is inadequate for structured descriptions.

Many picture processing systems whose chief component is a language for describing pictures have been reported in the literature. Most of these linguistic models deal with lines as terminal elements and use specialized cases of general set-replacement rules. The systems of Anderson (6), Evans (7), Feder (8, 9) and Shaw (10) are the most generally comprehensive in this class and fit the general two-level model in the survey by Miller and Shaw (11). Pictures are described by a low level primitive (terminal element) description, supplemented by a hierarchic description which describes how primitives are grouped into higher level structures. Semantic components, such as variables, may be associated with a primitive or a hierarchic description. The hierarchic description is defined by a generative grammar which may be used to parse a given picture.

Limitations of the descriptive capability of these systems are a line pattern orientation and a lack of ability to express relationships between objects. Inefficiency is also present due to the fact that the set-replacement rules are implemented by top-down exhaustive set-partitioning techniques which are inherently slow.

Other surveys of linguistic models are given by Feder (12), Swain and Fu (13), and Narasimhan (14, 15). In this class the paper of Kirsch (16) is of historical interest, while Chang (17,18) and Knoke and Wiley (19) consider structure more explicitly. Lastly, in the same class, there is a body of work which has a higher emphasis on pictorial relationships: from Londe and Simons (20), Inselberg and Kline (21) and Inselberg (22), to Clowes (23) and Barrow and Popplestone (24).

Models which consider relations on sets and which use explicit graph representations of the relations have become, during the course of this research, generally acknowledged as necessary for picture description. In order to characterize types of relations (which can be expressed in a graph structure), we have defined and enumerated basic formal properties of binary relations in McCormick and Schwebel (25).

Raphael (26) and Childs (27, 28) employ set-theoretic data structures with set operations; in addition Elliot (29) uses explicit properties of binary relations in a fact retrieval system. However,all three treat relations as sets

rather than as graphs. Eastman (30, 31) finds a need for a better spatial relationship descriptive capability. Relational networks, i.e. where a graph expresses the relations, are used in Savitt, Love, and Troop (32) and are considered as constraints for picture processing in Montanari (33). Finally, surveys of data structures for computer graphics can be found in Williams (34) and Gray (35).

1.3.3 Descriptive Pattern Analysis

Here we consider the methods of analysis which attempt to modify the initial picture representation so as to obtain the (usually reduced) desired description. This is the heart of the pattern recognition problem. In this class we will first consider heuristic object-formation techniques and model-driven analyzers. Then we consider graph-theoretic, graph-grammatical, graph-transformational, and structural inference techniques which have more pertinence to the present work. Surveys will be found in Evans (36), Duda and Hart (37), and Lipkin, Watt, and Kirsch (38).

The programs of Guzman (39, 40) demonstrated the feasibility of what we have called name-independent parsing of objects. Heuristics involving two-dimensional vertices were used to merge planar regions into three-dimensional objects using an abstract graph representation. We have been able to show that Guzman's work is conveniently expressible in our proposed model. In this case, planar faces form basic cells,

and vertex types imply relationships between cells. The composite (object) formation process of cell merging can be implemented by simple graph transformations with two classes of relations.

Falk (41) considers an analysis similar to Guzman's, but with imperfect data. Huffman (42, 43) is able to express more semantics in his interesting application of analogous vertex rules which are used to form polyhedra. Brice and Fennema (44) apply region-merging heuristics.

Decomposition in terms of mathematically-defined entities can be found in Pavlidis (45, 46) where primary sets are used to generate polygons, and in Michalski and McCormick (47), Michalski (48) and Raulefs (49) where interval covers are considered.

Model-oriented analyzers attempt to match a prototype to an instance of a picture. In Guzman (50, 51), rigid prototypes and models are used. More flexibility is obtained in Preparata and Ray (52) by using confidence measures of plausibility between the picture and the model. The structure and pattern matching techniques considered in Salton and Sussenguth (53) and Freeman and Gardner (54) are of interest for the prototype-instance match.

Graph-theoretic clustering techniques operate on relational graphs, attempting to merge nodes according to the relations between pairs of nodes. Matula (55) and Zahn (56) consider clustering on graphs using simple values (weights)

associated with unirelational graphs.

Graph-grammatical formulations consist of grammatical rewriting rules on labeled graphs. In Pflantz (57), convexity of graphs is considered as a preliminary to formulating web grammars in Pflantz and Rosenfeld (58). Montanari (59) considers contextual conditions to decide if a web-grammatical rule can be applied. A simple case of the embedding of a relation during application of a web rule is defined. Pavlidis (60) develops further theoretical results on graph grammars. Graph grammars are of great interest since they are a natural generalization of string grammars and offer hope that analogies to theoretical results of language theory can be shown.

Analysis techniques to infer or determine structure are considered in Kirsch (61) and Evans (62, 63). For this analysis multirelational graph structures are needed. Winston (64) shows how a structured description may be inferred from examples.

Explicit studies of graph transformations appear necessary for structural inference. Transformations on program graphs are considered in Cooper (65) and Basu (66). Our work in this area was initiated by McCormick (67) by presenting illustrative transformations and his emphasis upon the concept of reversibility. These ideas were further considered in Schwebel (68, 69) and led to definitions of properties of relations for composite formation in Schwebel and McCormick (70).

1.3.4 Processing Languages

Languages to implement picture processing algorithms are divided into two categories: descriptive graphic languages and graph or graph-structure processing languages.

Descriptive graphic languages have tended to be display-oriented and of limited use in recognition of pictures. In this class are the systems and languages of Herzog (71), Kulsrud (72) and Williams (73). In Schwebel (74), we presented criteria for graphic languages and defined a language, ICON, to meet these criteria. Still, the primary usefulness of ICON is for picture description.

Graph or graph-structure processing languages may be distinguished by the presence of operations which allow analysis of descriptions. Chase (75) uses a system for graph manipulation. Graph-structures of greater generality can be treated with languages given in Pratt and Friedman (76), Earley (77), Lieberman (78), Wolfberg (79), and Crespi-Reghizzi and Morpurgo (80). These languages are considered further in Chapter Five when we define our Structure Operation Language, SOL.

Associated with processing languages is the study of the transformations done on semantic information during processing. Means of propagating attribute values during application of processing rules are treated extensively in Knuth (81, 82) and considered in Milgram and Rosenfeld (83).

1.3.5 Non-Pictorial Applications

Graph structure representation and procedures are adequately general for immediate use in many applications outside of picture processing. We give only a few of them here.

Formulations of goal-directed general problem solving as in Ernst and Newell (84) and the generalized means-end analysis of Siklossy (85) can be expressed in a graph structure transformational framework.

Heuristic theorem proving applications that deal with structure can be found in Holden and Johnson (86), Slagle and Bursky (87), and Winston (88).

In medical diagnosis by sequential diagnosis of symptoms, Gorry (89) and Gorry and Barnett (90) implicitly use properties of attribute disease mappings between spaces for constructing disease patterns.

2. GRAPH STRUCTURES AND GRAPH-STRUCTURE TRANSFORMATIONS

2.1 Introduction

In this chapter we first give the basic definitions of a graph structure and of a graph-structure transformation. Some properties of transformations, including reversibility, are also defined. Examples are given to justify the definitions.

Then we undertake a logical study of graph transformations. Some simple graphs are enumerated to enable us to choose representative graph transformations. Simple embedding types of a relation under a transformation are defined and are then used to generate representative transformations from a root transformation. Finally, logical implications between transformations are shown.

2.2 Basic Definitions

2.2.1 Graph-Structure Representation

We first give the definition of a graph structure: The elements of a graph structure, GS, are represented by a sixtuple GS = $(\mathcal{N}, \mathcal{U}, \mathcal{B}, \mathcal{S}, \mathcal{G}, \mathcal{A})$ where

$\mathcal{N} = \{ n_1, n_2, \ldots n_m \}$ is a set of nodes

$\mathcal{U} = \{ u_1, u_2, \ldots u_r \}$ is a set of labels

$\mathcal{B} = \{ b_1, b_2, \ldots b_p \}$ is a set of labeled branches,

$b_i = (n_j, u_k, n_l)$, $\mathcal{B} \subseteq \mathcal{N} \times \mathcal{U} \times \mathcal{N}$.

$\mathcal{S} = \{s_1, s_2, \ldots s_w\}$ is a set of nodes and branches, $\mathcal{S} \subseteq 2^{\mathcal{N} \cup \mathcal{B}}$, $s_i \subseteq \mathcal{N} \cup \mathcal{B}$

$\mathcal{G} = \{g_1, g_2, \ldots g_q\}$ is a set of directed labeled graphs, $\mathcal{G} \subseteq \mathcal{S}$, with the property that if $g_i = \{N_i, B_i\}$ then N_i contains all the nodes of the branches in B_i.

$\mathcal{A} = \{a_1, a_2, \ldots a_v\}$ is a set of attributes for nodes, branches, sets, or graphs. $\mathcal{A} = \mathcal{N}a \cup \mathcal{B}a \cup \mathcal{S}a \cup \mathcal{G}a$. Each a_i is a function $a_i : \mathcal{Y} \rightarrow \mathcal{V}_i$, where $\mathcal{Y} = \mathcal{N}$ or $\mathcal{B}$ or $\mathcal{S}$ or $\mathcal{G}$, and $\mathcal{V}_i$ is a set of values.

Equivalently, we may consider the branches as a set of relations $\{R_{u_1}, R_{u_2}, \ldots R_{u_r}\} = \{R_u\}$ where $R_u \subseteq \mathcal{N} \times \mathcal{N}$, and a branch $b_i = (n_j, u_k, n_l)$ exists if and only if $(n_j, n_l) \in R_{u_k}$. In this case each relation corresponds to a directed graph.

Or, we may expand the idea of a relation by associating a weight, $w \in \mathcal{B}a$, with each branch. Then we consider R_u to be a function, $R_u : \mathcal{N} \times \mathcal{N} \rightarrow w'$, with a value from the set consisting of weights w and a null weight λ, $w' = w \cup \lambda$. Each relation corresponds to a set of weighted branches of a digraph. The null weight represents a missing branch.

Definition: g_i is a subgraph of g_j if and only if $g_i \in \mathcal{G}$, $g_j \in \mathcal{G}$, and $g_i \subseteq g_j$.

The following available functions will be defined to operate on elements of graph structures:

TAIL: $\mathcal{B} \rightarrow \mathcal{N}$, $\text{TAIL}(n_j, u_k, n_l) = n_j$

HEAD: $\mathcal{B} \rightarrow \mathcal{N}$, $\text{HEAD}(n_j, u_k, n_l) = n_l$

$$\text{LABEL}: \mathcal{B} \rightarrow \mathcal{U}, \quad \text{LABEL}(n_j, u_k, n_l) = u_k$$
$$\text{NODES}: \mathcal{S} \rightarrow 2^{\mathcal{N}}, \quad \text{NODES}(s_i) = s_i \cap \mathcal{N}$$
$$\text{BRANCHES}: \mathcal{S} \rightarrow 2^{\mathcal{B}}, \quad \text{BRANCHES}(s_i) = s_i \cap \mathcal{B}$$

A graph structure is, thus, a set of nodes and labeled branches, with identifiable (sets of) graphs, subgraphs, and sets of nodes and branches. There is also a set of associated attributes which map from a set of graph-structure elements of a given type into a set of values.

Graph-structured data representation and graph-transformational-based parsing procedures are basic to our model.

In our picture processing applications, a graph structure has the following semantics. Nodes represent primitives and higher level derived composite elements. Labeled branches represent binary relations between objects. Graphs and subgraphs usually represent the local domain or context of a transformation. Sets of nodes and branches may represent regions, paths, cut sets, etc. Much of the semantics of a particular application is carried in the attributes of nodes, branches, graphs, or sets. For example, a primitive node can be associated with the physical picture data which it represents.

2.2.2 Graph-Structure Transformations

A graph-structure transformation, T, between two structures L_1 and L_2 is a relation $T \subseteq g_1' \times g_2'$ where $g' = g \cup 2$,

$g_1 \in G_1$, $g_2 \in G_2$. Here we assume an element, λ, called the empty element, which is associated with each graph, g. An element of g_1, mapped into λ_2 only, is said to be deleted by the transformation, and an element of g_2 which is the image of λ_1 only, is said to be created by the transformation.

The inverse of a graph-structure transformation T is denoted T^{-1} and is the inverse of the relation. That is, $T^{-1} = \{(y, x) \mid (x, y) \in T\}$.

Note. Delete is the inverse of create.

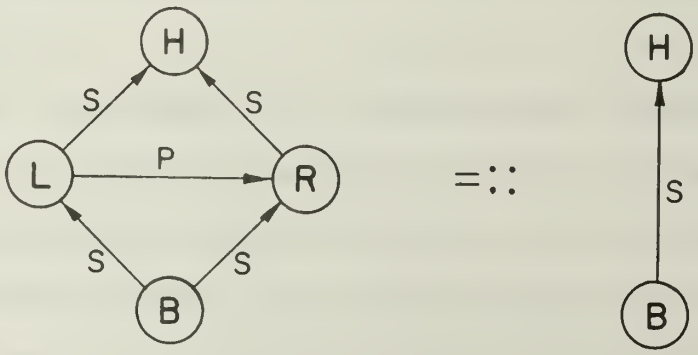
A graph-structure transformation will also be represented by the rule $g_1 :: g_2$ which specifies that the structure g_2 will replace the structure g_1. The relation T may either be implicitly understood or explicitly stated.

Below we give some properties of transformations and the reasons for defining graph transformations with these properties.

A graph transformation is a relation. Transformations specify a change in a graph structure to produce a new graph structure. In general, we want to allow contracting ($|g_1| > |g_2|$) and expanding ($|g_2| > |g_1|$) transformations. Many-to-one as well as one-to-many transformations can occur. For example, even in a composite-forming (usually many-to-one) transformation in picture processing, an edge may be subdivided between two regions. Because we wish to allow contextual elements to be included in the domain or range of the

transformation, the relation need not be everywhere-defined or onto. We also wish the inverse of a transformation to be a well-defined transformation.

Transformations act on a graph (nodes and branches) and produce a graph (nodes and branches). We wish the domain to be a related set, in effect a substructure specified by a graph and its associated semantics. A transformation could be restricted to map nodes into nodes, or nodes and branches into nodes. These restrictions are not adhered to here, since the former choice neglects the information contained in the relations, and the latter produces a non-invertable transformation which also neglects relations in the produced graph structure. Figure 1 gives an example of a transformation in which a branch has both nodes and branches in the inverse image.



H = head, B = base, S = supports, P = parallel to, L = left brace
R = right brace, T(H) = H, T(B) = B, T(L) = T(R) =
T((L, P, R)) = (B, S, H)

Figure 1. A Transformation Which Maps Nodes and Branches into a Branch

Transformations map <u>elements of a graph to elements of a graph</u>. This allows us to associate elements differently even though the graphs g_1 and g_2 are the same. Thus in Figure 2, the three transformations T_1, T_2, T_3 relate different elements of the graph g_1' to the graph g_2'.

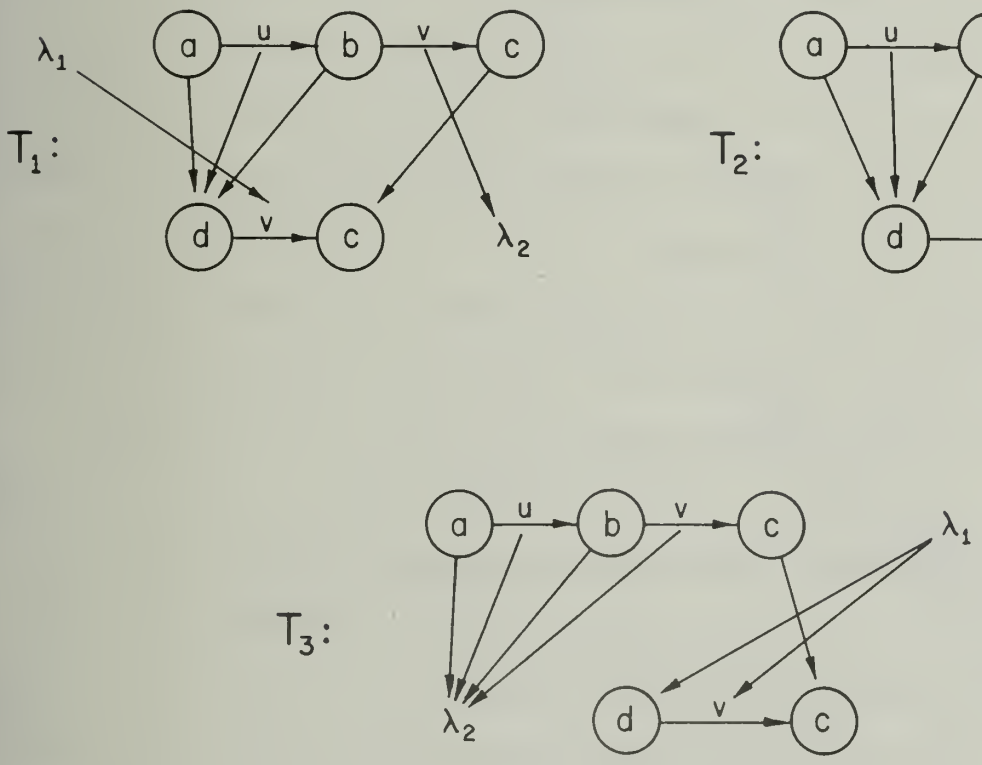


Figure 2. Three Different Transformations Between Identical Graphs

Deletions and creations of elements may be specified. The empty element, λ, is introduced for this purpose. We could have defined unmapped elements to be deleted, but that would have eliminated the possibility of having elements in the graph which are present only to establish context and which are not mapped or deleted.

The element mapping and the presence of the empty element in a transformation also allow us to define the memory of an element in the range of a transformation. This is a useful concept to have when considering conditions for performing the inverse transformation and to simplify the representation of a series of graph transformations. The memory, denoted M(e), of an element e which is in the range of the transformation T_j is the inverse image of e under T_j, that is, $M(e) = T_j^{-1}(e)$. Normally in a series of transformations, T_j will be the last transformation which had e in its range.

Then after a transformation, $T \subseteq g_1' \times g_2'$, we have available the transformed graph structure L_2, which contains the range of T, and the history of the transformation, represented by the memory of the elements of g_2: M(e) for $e \in g_2$, $e \neq \lambda_2$. This simplifies the representation of g_2, which will then be the domain for further transformations. The memory of the empty element is defined to be empty, $M(\lambda) = \emptyset$. Thus, no history is maintained for deleted elements. For example, in Figure 2 under transformation T_1, d has three elements in its memory due to T_1: $M(d) = \{a,(a,u,b),b\}$. Under T_3, d has an empty memory.

Figure 3 shows a transformation between structures L_1 and L_2. The graphs, g_1 and g_2, are encircled and are referred to as the domain balloon and the range (or transformed) balloon, respectively, of T. Branches A and B cut the balloon boundaries. The element h and the memory of h, M(h), are shown cross-hatched in the figure.

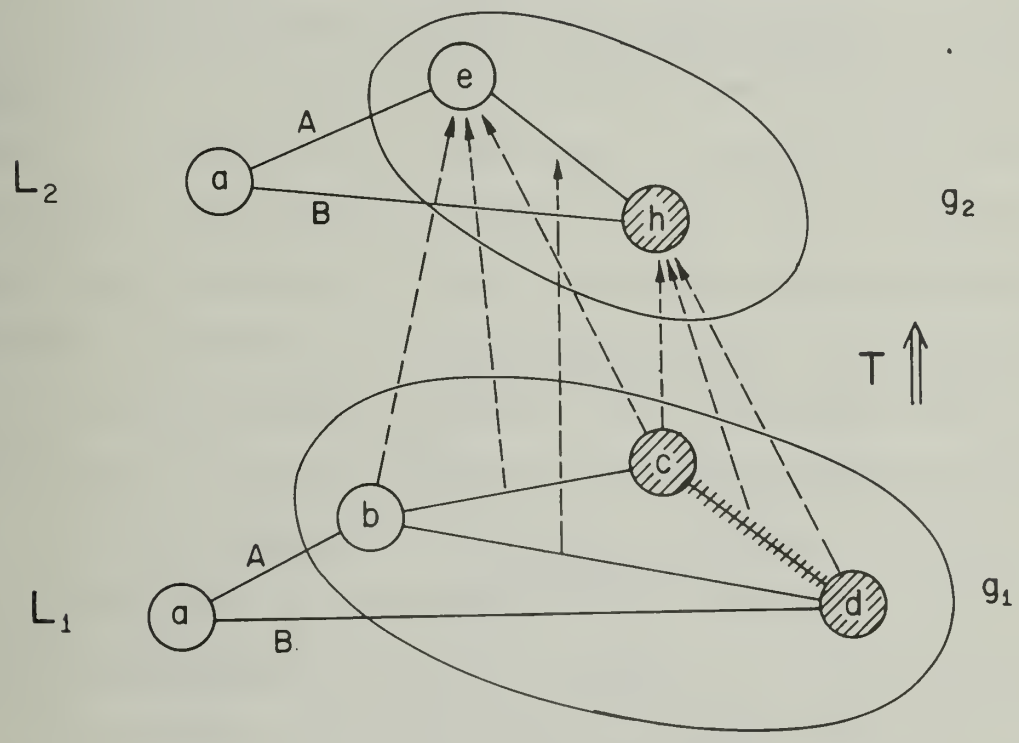


Figure 3 . Illustration of Transformation Balloons and Memory

Consider a graph transformation T as a grammatical rewriting rule for graph structures. Here T will be specified in terms of variables in η and β e.g. T_2(A, B, a, b, c) where

A, B $\in \mathcal{B}$ and a, b, c $\in \mathcal{N}$. We will be concerned with the case in which the rewriting rules are used to reduce a graph structure to a simpler form. These rules will be applied in graph-structure parsing procedures. Thus, we generally want $|g_1| > |g_2|$. In many cases, T will be single-valued and will not create new elements, T: $g_1 \rightarrow g_2'$.

2.2.3 Reversibility of Graph Transformations

An application of a transformation is valid if the resulting graph structure is valid. This will depend upon the logical definitions of the system. For example, if composites represent unions of set elements, and branches represent binary relations between elements, then the validity of the transformation depends upon the validity of the relations created by the transformation. When the transformation contains variables for nodes, branches, or attributes, we will say that the transformation is valid over some set of variable values.

Assuming a set of values for the variables in a transformation , we will say that T is reversible if and only if T is valid and T^{-1} is valid, given the memory of T. A reversible transformation will be said to be information lossless. A reversible transformation is one which can be validly inverted assuming the memory of the transformation is available.

In parsing an initial graph structure, information lossless transformations are plausible choices to use in order to infer composites. In general, this criterion can be expanded

so that each transformation may have a degree of reversibility associated with it. A transformation with a high degree of reversibility can produce a structure close (in the sense of "near-miss," see Chapter Four) to the given structure.

Example: Figure 4 shows transformations S and T,

$A = a_1 \cup a_2$, $B = b_1 \cup b_2$, $1 = a_1 \cup b_1$ and $2 = a_2 \cup b_2$.

From the properties of the relations, "Above" and "Inside," we can conclude that T is reversible, while S is not reversible, for any variables a_i and b_j from the Boolean algebra of picture subparts.

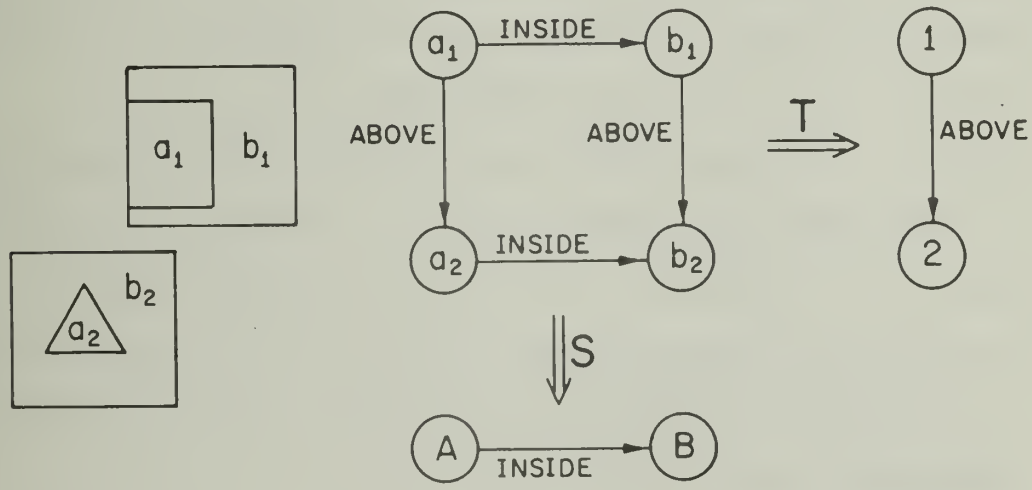


Figure 4. Example of Reversible and Irreversible Transformations

2.3 Logical Study of Graph-Structure Transformations

2.3.1 Representative Transformations

2.3.1.1 Operators on Relational Systems

In general, we may consider graph structures as relational systems, $S = \{R_u\}$, on one set or between sets, $R_u \subseteq M \times N$. We define a group Z of operators, z, which act on relational systems to generate other relational systems, denoted z(S). The operators also generate statements about z(S) from statements about S.

The first operators considered form inverses of the relations. Then, when a lattice structure is assumed on the sets, duality operators are introduced. Thus, while considering and proving statements about a system S, the operators, $z \in Z$, allow us to imply similar statements about systems with inverse relations and/or dual sets.

First we consider a set of relations $\{R_{u_1}, R_{u_2}, \ldots R_{u_r}\}$. In this case, an inverse operator is defined for each u. The group Z of operators is generated by composition of the inverse operators defined in Table 1 where z is an operator in Z and A is a logical expression in the system such as a graph or transformation.

Table 1 . Generators of the Group Z

z	z(A)
E	A
I_u	A with R_u replaced by R_u^{-1}
I	$I_{u_1} \; I_{u_2} \ldots I_{u_r}$

The group Z is a commutative group with 2^r elements obtained from r different options of the inverse operators. In studying root transformations, we will restrict our attention to the case where r = 2 and $\mathcal{U} = \{u_1, u_2\}$, where the four operators in Z are:

E, I_1, I_2, I Here, $I = I_1 I_2$.

2.3.1.2 Enumeration of Basic Graphs

In order to choose some representative labeled graphs as operands for the domain and range of transformations, we will enumerate graphs with up to four points, two relations, and four edges, i.e. $m \leq 4$, $r \leq 2$, $p \leq 4$;

(i) Graphs which are isomorphic, i.e. identical except for a renaming of labels and points, are considered equivalent in the enumeration.

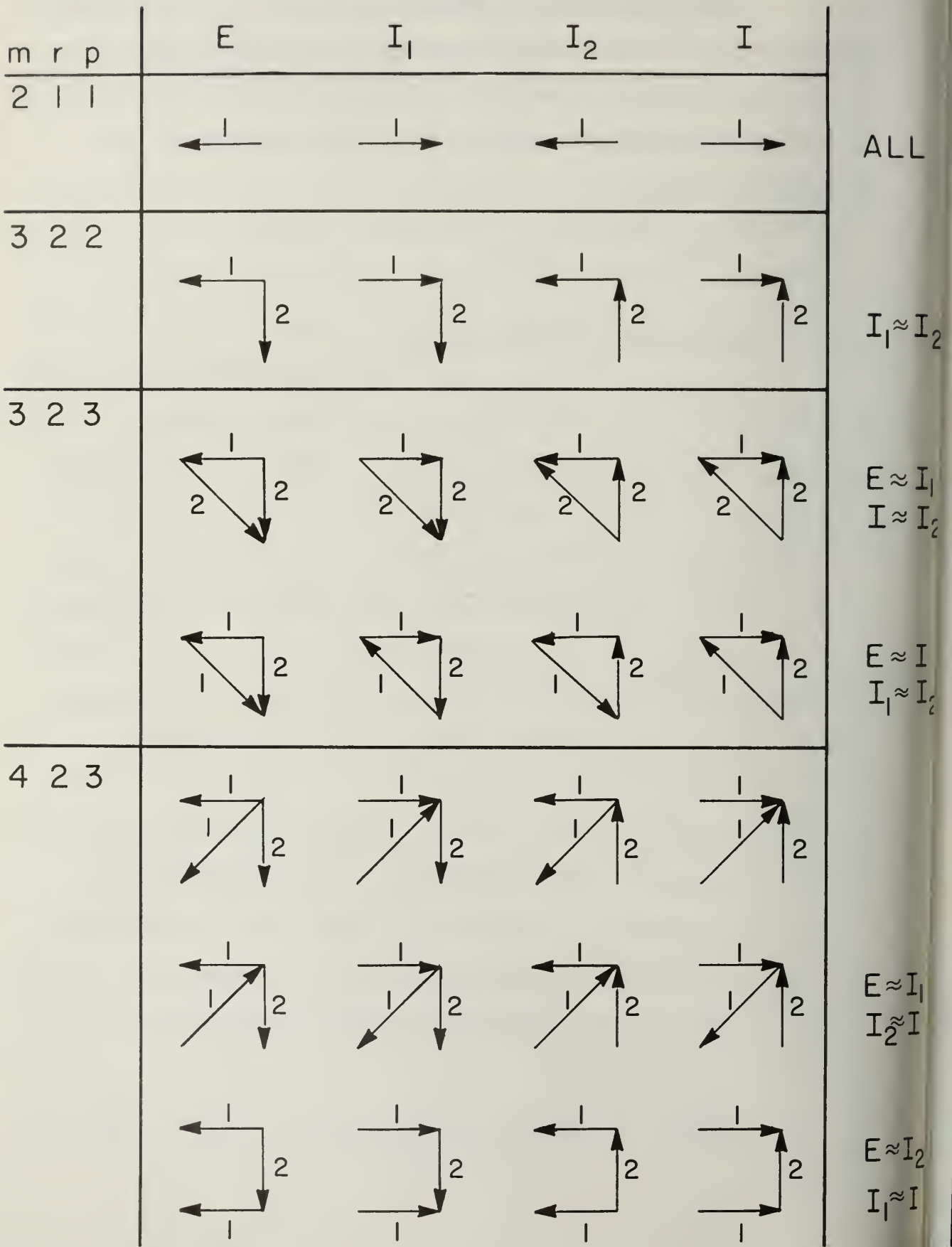
(ii) Graphs with reflexive loops will not be considered. Thus, edges will be exhibited between distinct points only and $m \geq 2$.

(iii) Multiple relations between two points will be treated as defining a new relation. Accordingly, graphs with multiple edges will not be enumerated.

(iv) Disconnected graphs are treated by considering their disconnected components only and are omitted in the enumeration.

Figure 5 displays the enumerated graphs. Each row

Figure 5a. Basic Graphs



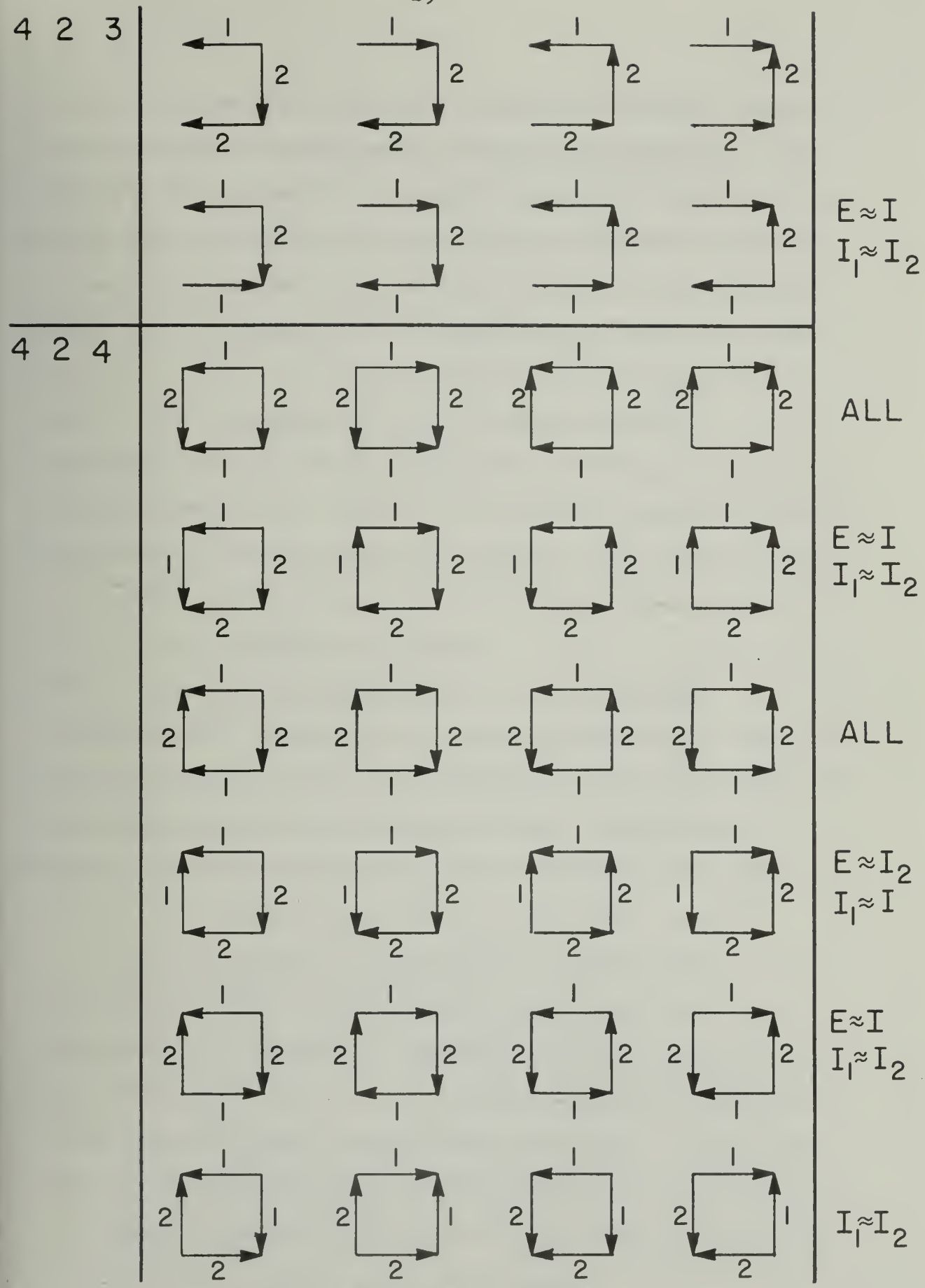


Figure 5b. Basic Graphs

contains a graph, L, and all the inverse graphs $I_1(L)$, $I_2(L)$, I(L). Isomorphisms may exist between graphs in the same row only. A list of isomorphic graphs in a row, or the word ALL is given on the same line to the right of the row. Thus, for example, all of the 2, 1, 1 graphs are isomorphic; in the first row of the 3, 2, 3 graphs, E is isomorphic to I_1, and I is isomorphic to I_2.

Except for the 2, 1, 1 case, graphs with r = 1 are not listed, since they are special cases of graphs with r = 2 which are listed. For m = 2 or 3 and r = 2, a graph with any possible value of p is equivalent to one of the listed graphs. For m = 4 and r = 2, the 4, 2, 1 and 4, 2, 2 graphs are disconnected. All 4, 2, 3 graphs are listed.

Additional 4, 2, 4 graphs may be obtained by adding an edge to any corner of the 3, 2, 3 graphs. These graphs are not listed since transformations will be applied to their 3, 2, 3 subgraphs. Similarly, other more complex graphs will be handled by a series of transformations on simpler subgraphs.

2.3.1.3 Definition of Representative Transformations

We choose three representative composite-forming transformations which use graphs in Figure 5 as operands. One graph is selected for each p = 2, 3, 4. The graphs chosen from Figure 5 are then generalized so that the branches may designate more than two relations. Instead of the two branch labels: u_1, u_2, we now use up to four labels: A, B, C, D.

The branch label E is used for the resulting branch. For descriptive purposes, when no ambiguity is possible, we sometimes use the branch label to denote a branch. Thus u_k may be used to denote the branch (n_j, u_k, n_1). We also assume that the operator I_1 acts on R_A and R_D, and that the operator I_2 acts on R_B, R_C, and R_E. This will allow more generality in this section while keeping the operator group simple. The choice of the A, D, and B, C, E groupings, under I_1 and I_2, respectively, is justified later, in connection with feasible root transformations. Figure 6 displays the three transformations Q2, Q3, Q4 and also z(Q2), z(Q3), z(Q4) for the four z in Z. There are a total of seven non-isomorphic transformations generated by Q2, Q3, and Q4.

The transformation Q2 is specified so that the branch B is deleted and the branch E is created, while a, A, and b are all mapped into the node e. The memory of e is then available, along with the transformed graph, for the application of further transformations. Q3 and Q4 are defined similarly.

The transformation mappings are given below:

$$Q2(a) = Q2(b) = Q2(A) = e,$$
$$Q2(B) = \lambda_2, \quad Q2(c) = c, \quad Q2(\lambda_1) = E,$$
$$\text{thus } M(e) = \{a, A, b\}, \quad M(E) = \emptyset, \quad M(c) = c.$$

$$Q3(a) = Q3(A) = Q3(b) = e,$$
$$Q3(C) = Q3(B) = \lambda_2,$$
$$Q3(c) = c, \quad Q3(\lambda_1) = E.$$

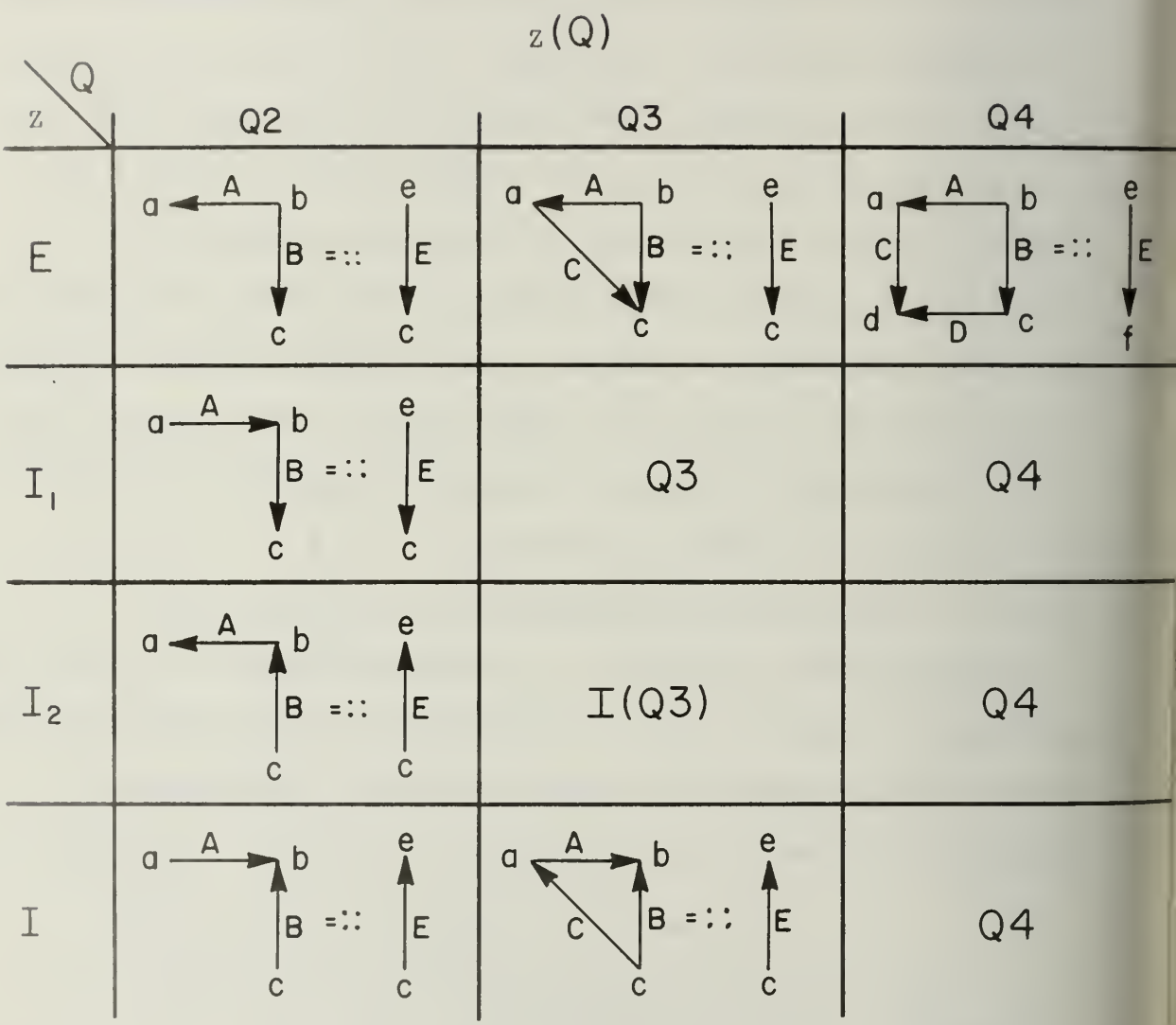


Figure 6. Three Representative Transformations and Generated Transformations

$$Q4(a) = Q4(A) = Q4(b) = e,$$
$$Q4(d) = Q4(D) = Q4(c) = f,$$
$$Q4(C) = Q4(B) = \lambda_2,\ Q4(\lambda_1) = E.$$

The basic criteria for the choice of transformations were simplicity and plausibility of composite formation. It is assumed that simple transformations, such as Q2, will occur most frequently and will be basic to any application procedure. More complex transformations are selected if one of the ways in which composites could be chosen appears intuitively most reasonable. For example, of the two basic 3, 2, 3 graphs in Figure 5 , the one selected for Q3 gives an obvious choice of composite formation assuming that a relation can be extended to a composite if it holds with all parts of the composite.

2.3.2 Embedding Types and Root Transformations

2.3.2.1 Simple Embedding Types

We first consider general rules for embedding a graph g_2, which has resulted from a transformation $T \subseteq g_1' \times g_2'$, into the graph structure of g_1. These rules are considered as properties of relations for arbitrary transformations, and are called simple embedding types of the relation. The embedding rules determine only whether a branch (n, u, m), where n is outside of and m is within the domain balloon, implies a new branch (n, u, t) where t is within the transformed balloon. The decision as to whether a branch exists depends upon the inverse image, $T^{-1}(t)$, of the resulting element.

Three logical embedding types are defined below. Assume $n, m \in \mathcal{N}$, $n \notin g_1$, $m \in g_1$ $u \in \mathcal{U}$, $t \in T(m)$, $t \neq \lambda$.

Simple Embedding Type

OR	(n,u, m) For any $m \in T^{-1}(t)$	$\Rightarrow$ (n, u, t)
AND	(n, u,m) For all $m \in T^{-1}(t)$	$\Rightarrow$ (n, u, t)
NOT	(n, u,m) For no $m \in T^{-1}(t)$	$\Rightarrow$ (n, u, t)

Thus, embedding types allow us to determine connections from an element outside the domain balloon to the elements within the transformed balloon as logical functions of the elements of the inverse image of the new element.

Let NODES($T^{-1}(t)$) = $\{m_1, m_2, \ldots m_k\}$ and b = (n, u, t), b_i = (n, u, m_i), as shown in Figure 7 . Let f_i be a predicate which is true if and only if branch b_i exists; similarly, f is a predicate which is true if and only if the branch b exists. Then the embedding types, OR, AND, and NOT are expressed in terms of predicates as given below.

$$\text{OR} \quad \bigvee_i f_i \Rightarrow f$$

$$\text{AND} \quad \bigwedge_i f_i \Rightarrow f$$

$$\text{NOT} \quad \neg\left(\bigvee_i f_i\right) \Rightarrow f$$

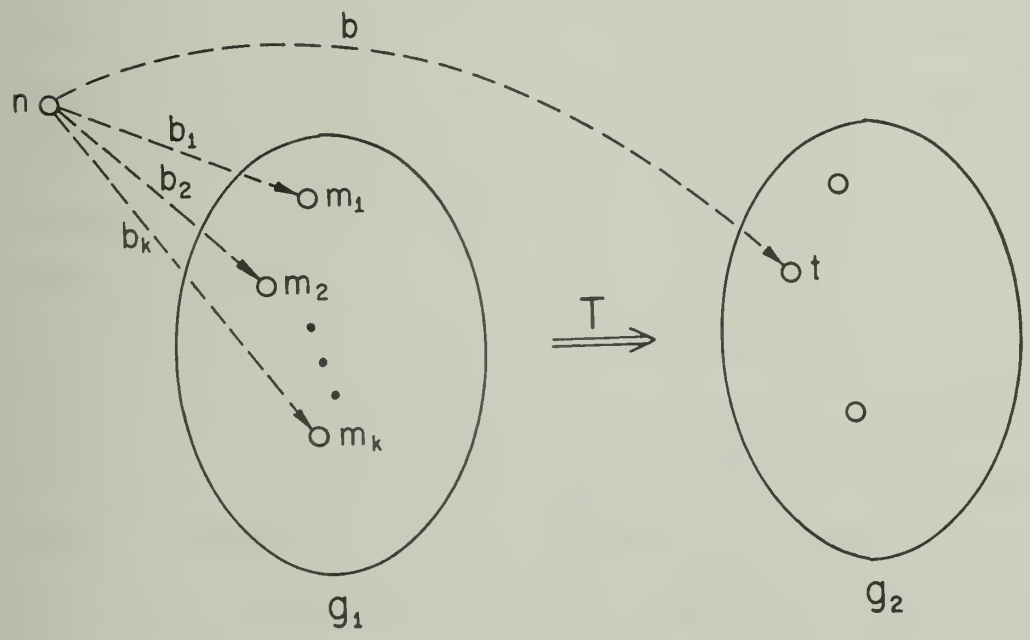


Figure 7 . Simple Embedding Types

2.3.2.2 Root Transformations

We wish to study the application of a set of simple root transformations in order to find:

-when a set of simple transformations and associated embedding types implies more complex transformations.

-the set of transformations which may be generated by a root set.

Consider again the simple transformations Q2, Q3, Q4 of Figure 6. First assume a root transformation RT_1: $\{a, (a, A, b), b\} \longrightarrow e$. Thus, RT_1 maps two nodes and a branch between them into a single node.

Now we wish to apply RT_1 to the domain of Q2, Q3, Q4 in order to effect these transformations. It is necessary to specify the embedding of the branches connecting a and b. The conditions for implementing Q2, Q3, and Q4 by means of root transformations, RT_1, specifying an OR type embedding for all the branches, are shown in Figure 8. The condition for Q2 to be implemented by RT_1 is that $R_E = R_B$. If this condition holds, we call the transformation a new basic root transformation RQ2. Similarly for Q3 and Q4, if $R_E = R_C \wedge R_B$ then we have basic root transformations RQ3 and RQ4. Figure 9 shows RQ2, RQ3 and RQ4. The conditions are now expressed in the graphs by having the same branch label, B, wherever B, C, or E appeared in Q2, Q3, and Q4.

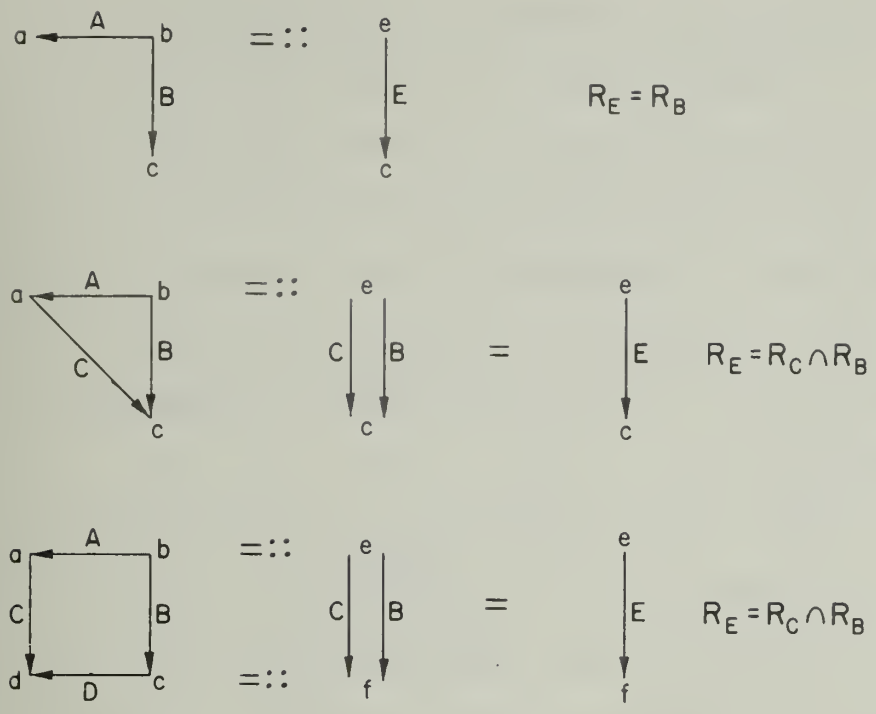


Figure 8. Representative Transformations

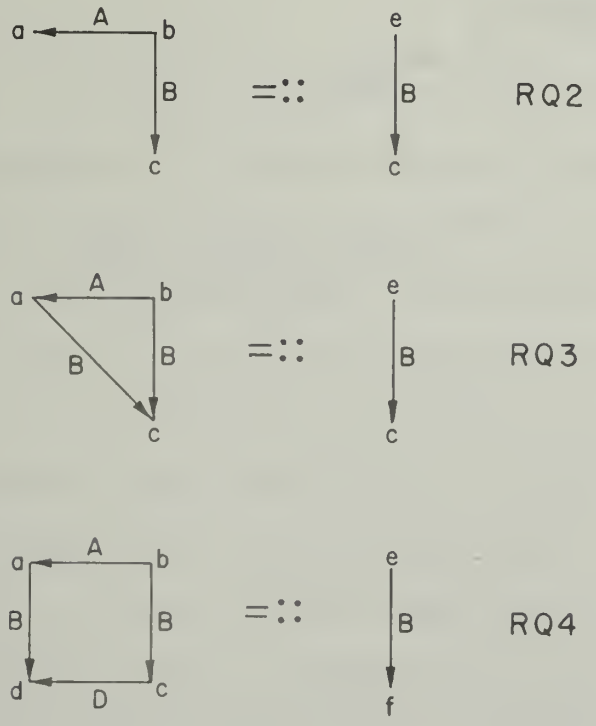


Figure 9. Generated Root Transformations

2.3.2.3 Implications Between Transformations

In order to aid in the establishment of sufficient conditions for the validity of transformations, we will first derive implications between the transformations RQ2, RQ3, and RQ4. The group of operators, Z, will be used to simplify the derivation. In particular, a logical statement, A, implies a transformation, Q, if and only if z(A) implies z(Q), where z is an operator in Z.

Again it is necessary to specify embeddings. Since deletion of branches not named in the transformation may eliminate necessary constraints, we will assume an OR type embedding for branches so that the maximum information is maintained. The following implications then hold:

$$RQ2 \implies RQ3$$

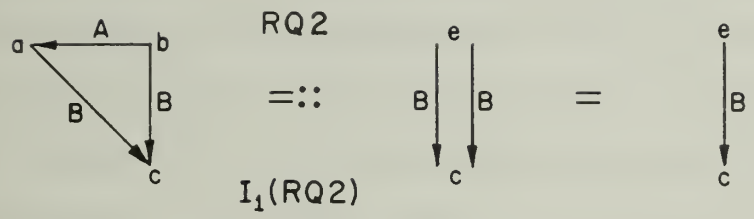
$$I_1(RQ2) \implies RQ3$$

$$RQ2 \wedge I(RQ3) \implies RQ4 \quad \text{if } R_D = R_A$$

Figure 10 shows diagrams to establish these implications.

The variables of a transformation may be explicitly stated by a parenthesized list following the transformation name. The variables may be omitted if they range over all possible values in the set. Let RQ2(A, B, a, b, c), RQ3(A,B,a,b,c) and RQ4(A, B, D, a, b, c, d) be standard orders for representing the variables in the transformations of Figure 9 . If a variable is omitted commas may be left out at the right end of the string. With this notation we may specify a less strict condition for RQ4:

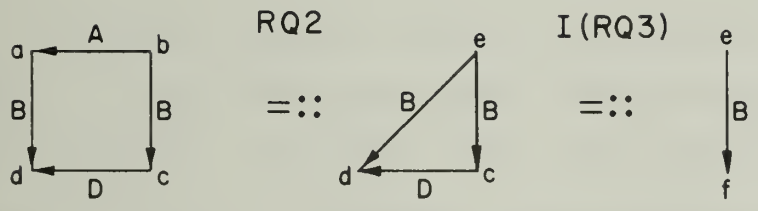
$$RQ2(A, B) \wedge I(RQ3(D, B)) \implies RQ4(A, B, D)$$



$I_1(RQ2)$

$RQ2 \Longrightarrow RQ3$

$I_1(RQ2) \Rightarrow RQ3$



$RQ2 \wedge I(RQ3) \Longrightarrow RQ4$

if $R_D = R_A$

Figure 10. Implications Between Root Transformations

3. PROPERTIES OF COMPOSITE-FORMING TRANSFORMATIONS

3.1 Introduction

In this chapter we consider composite-forming graph transformations. We wish, eventually, to consider a series of transformations which reduce graphs to single nodes. Chapter Two considered simple embeddings between only one node and its transformed environment. When the nodes of the original graph are partitioned and coalesced into more than one node, we want to infer relations between composite nodes from the previously defined relations between the member nodes, which belonged to different inverse graphs of the new composites. This latter type of embedding is called a composite embedding type.

Composite embedding types are defined in section 3.2. In that section results are exhibited only for series of transformations which can be expressed equivalently as a pair of transformations, each producing one node. This assures that the transformations are composite-forming and allows us to develop implications between simple embedding types and composite embedding types.

It is obvious that the concept of correspondences (binary relations) between two sets is very general and can be employed practically in algorithms only when specialized cases are considered. We will specialize in two ways. First the sets considered will be assumed to have some structure. An

example of this structure is the Boolean algebra of picture parts formed by the relationship "is a subpart of." Secondly, the binary relations between the sets will be categorized with respect to their invariance under composite-forming transformations. For example, the image processing relation "above", between picture subparts, can be extended to composite picture elements which are formed by the union of the subparts. The composite formation property used here is the following:

a "above" b and a "above" c implies a "above" (b union c),

where a, b, and c are arbitrary picture elements. This implication gives a (partial) formal characterization of the binary relation "above"- by defining one of its properties under composite formation.

Section 3.3 imposes conditions on the transformations to develop more properties of relations in specific transformation systems. The structure of the sets is here defined to be a lattice, and systems consisting of relations between lattices are defined; the composite-forming properties defined are called lattice operational properties. Here the transformations are restricted to mappings which can be expressed by lattice operations, for example, as the join of a pair of elements. The restriction of transformations will aid in the description and decomposition of a series of T's, studied in Chapter Four.

In Section 3.4, the simple embedding types, composite embedding types, and lattice operational properties are considered together. Root transformations, RQ's, as defined in Chapter Two,are now specialized to lattice operations and called LRQ's.

The relationships between the LRQ's and the lattice operational properties are shown.

3.2 Composite Embedding Types

3.2.1 Preliminary Remarks

We wish to consider the result of a series of transformations which map a graph with nodes $\{a_i\}$ into a single node a and a graph with nodes $\{b_j\}$ into a single node b. In general, we wish to be able to determine the relations between a and b from the relations among the a_i's and b_j's.

Figure 11 shows the domain and range of the transformation T. In general T will be a series of functional, i.e. single-valued, transformations, $T = T_F, \ldots T_2 \cdot T_1$. We assume that each T_i is functional and also that T is functional so that $T^{-1}(a) \cap T^{-1}(b) = \emptyset$.

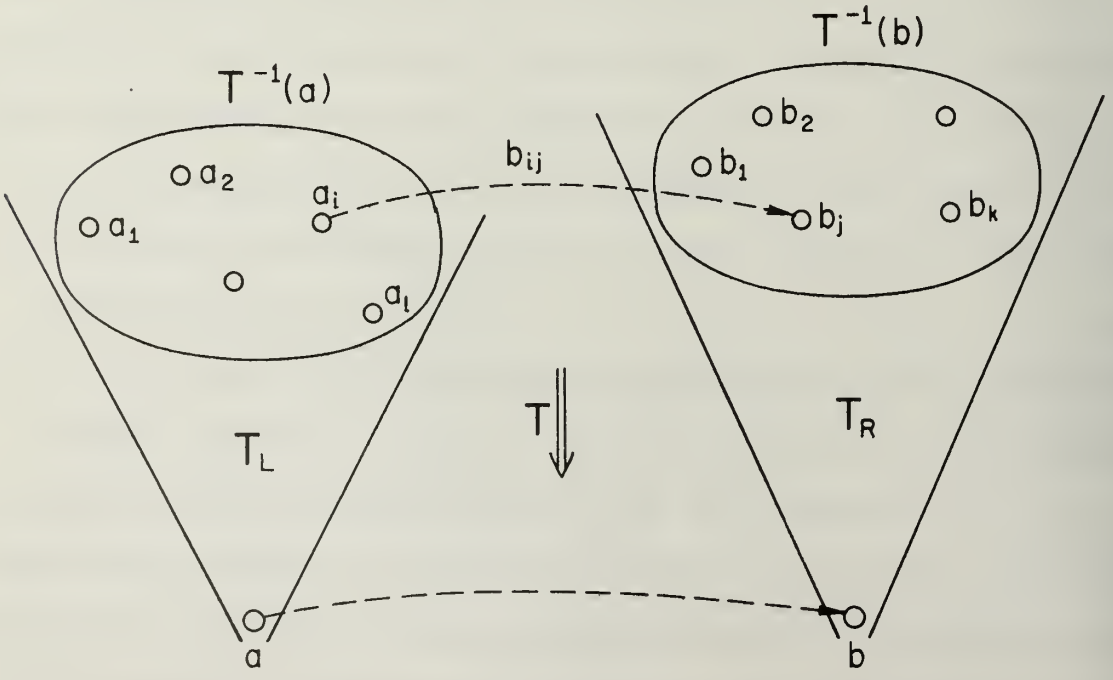


Figure 11.Illustration for Composite Embedding Types

We want to use the relation embedding types which were defined for a single transformation to imply the existence of aR_ub from the instances of $a_iR_ub_j$.

For a series of transformations the order of application will be important. However, the restriction to functional transformations has eliminated the interaction between $T^{-1}(a)$ and $T^{-1}(b)$ so that we may assume $T = T_L \cdot T_R$ or $T_R \cdot T_L$, where $T_L = T_{L_M} \ldots T_{L_2} \cdot T_{L_1}$ and $T_R = T_{R_N} \ldots T_{R_2} \cdot T_{R_1}$. T_L acts on $T^{-1}(a)$ and T_R acts on $T^{-1}(b)$.

In some cases, to prove implications about the composite embeddings, aR_ub, from the simple embedding types of R_u, one order, $T_R \cdot T_L$ or $T_L \cdot T_R$ will be preferred. T_R will use the simple embedding type of R_u, and T_L will use the simple embedding type of R_u^{-1} to determine intermediate embeddings. After defining some composite embedding types, CET's, for a relation R_u, we will show which of them can be implied by the embedding types of R_u and R_u^{-1}.

3.2.2 Definition of Composite Embedding Types

Definition. R_u has Composite Embedding Type K, denoted CET K, if and only if $a_iR_ub_j$ for all $(i,j) \in \Delta_k \Rightarrow aR_ub$, where $\Delta_k \subseteq I \times J$. Thus, CET K is defined by specifying Δ_k.

Below we specify Δ_1, Δ_2, Δ_3, Δ_4, Δ_5, and Δ_6 to specify six basic composite embedding types.

CET 1 "ANY"

$\Delta_1 \neq \emptyset$ i.e. there exists $a_i R_u b_j$

CET 2 "ALL-TAILS"

$\Delta_2 \supseteq \{(i, j(i)) | \forall i\}$ i.e. for all i there exists j(i) such that $a_i R_u b_{j(i)}$

CET 3 "ALL-HEADS"

$\Delta_3 \supseteq \{(i(j), j) | \forall j\}$ i.e. for all j there exists i(j) such that $a_{i(j)} R_u b_j$

CET 4 "ALL-ONE"

$\Delta_4 \supseteq I \times j_p$ i.e. for some j_p, $a_i R_u b_{j_p}$ for all i

CET 5 "ONE-ALL"

$\Delta_5 \supseteq i_p \times J$ i.e. for some i_p, $a_{i_p} R_u b_j$ for all j

CET 6 "ALL"

$\Delta_6 = I \times J$ i.e. $a_i R_u b_j$ for all i and j

Let f_{ij} be a predicate which is true if and only if branch $b_{ij} = (a_i, u \cdot b_j)$ exists, and let f be a predicate which is true if and only if the branch $aR_u b$ exists. Then we can define the composite embedding types in terms of the branch predicates as given below.

CET		
1	ANY	$\bigvee_{i,j} f_{ij} \Rightarrow f$
2	ALL-TAILS	$\bigwedge_{i}(\bigvee_{j} f_{ij}) \Rightarrow f$
3	ALL-HEADS	$\bigwedge_{j}(\bigvee_{i} f_{ij}) \Rightarrow f$
4	ALL-ONE	$\bigvee_{j}(\bigwedge_{i} f_{ij}) \Rightarrow f$
5	ONE-ALL	$\bigvee_{i}(\bigwedge_{j} f_{ij}) \Rightarrow f$
6	ALL	$\bigwedge_{i,j} (f_{ij}) \Rightarrow f$

3.2.3 Implications Between Embedding Types

Figure 12 shows implications between the composite embedding types by an implication lattice. The downward direction means "implies." The lattice is completed by taking logical "and's" and "or's" of the five basic types. For example, CET 2 & 3 could be defined by:

$$2\ \&\ 3:\quad (\bigwedge_{i}(\bigvee_{j} f_{ij})) \wedge (\bigwedge_{j}(\bigvee_{i} f_{ij})) \Rightarrow f$$

Also, we can see from the figure that:

CET 1 $\Rightarrow$ CET 2 $\Rightarrow$ CET 4 $\Rightarrow$ CET 6, and

CET 1 $\Rightarrow$ CET 3 $\Rightarrow$ CET 5 $\Rightarrow$ CET 6.

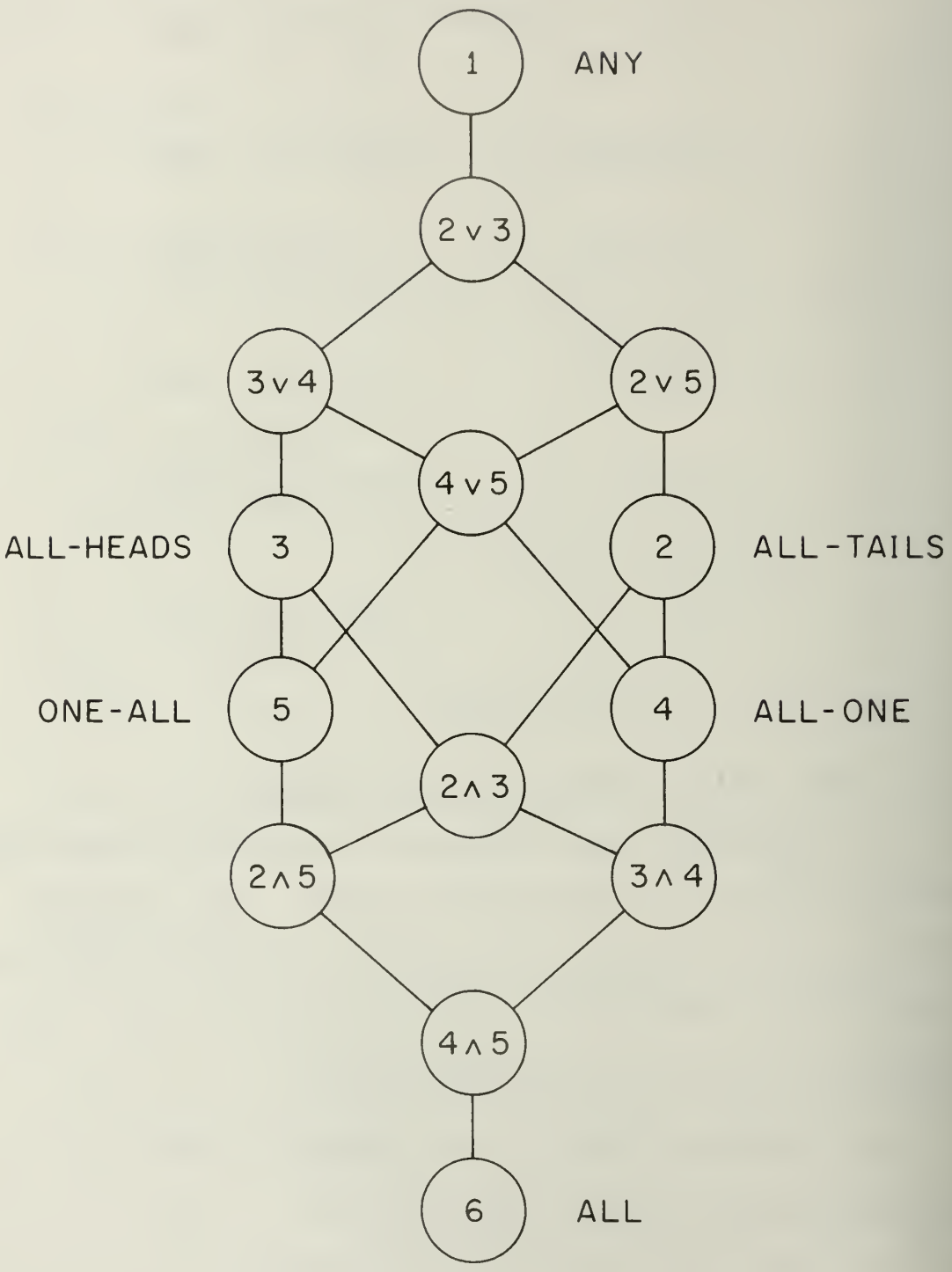


Figure 12. Implications Between Composite Embedding Types

The next theorem states implications between the simple embedding types of a relation R_u and R_u^{-1} and the composite embedding type of the relation R_u. The theorem is expressed by Table 2 given below. For example, the table states that if R_u has embedding type "OR" and R_u^{-1} has embedding type "AND," then R_u has composite embedding type "ALL-TAILS."

Table 2 . Theorem Relating Embedding Types

R_u has embedding type / and	OR	OR	AND	AND
R_u^{-1} has embedding type / implies	OR	AND	OR	AND
R_u has composite embedding type	ANY	ALL-TAILS	ALL-HEADS	ALL

Since we are considering the case where T is expressible as two independent transformations, $T = T_L \cdot T_R = T_R \cdot T_L$, the proof of the theorem is obvious by just considering the simple embedding type of R_u when performing T_R, and the simple embedding type of R_u^{-1} when performing T_L.

We illustrate by a picture processing example shown in Figure 13 . At the top of the figure is an abstract graph and a possible two-dimensional picture corresponding to the graph. Also shown are the transformations T_L and T_R which form the composites a and b.

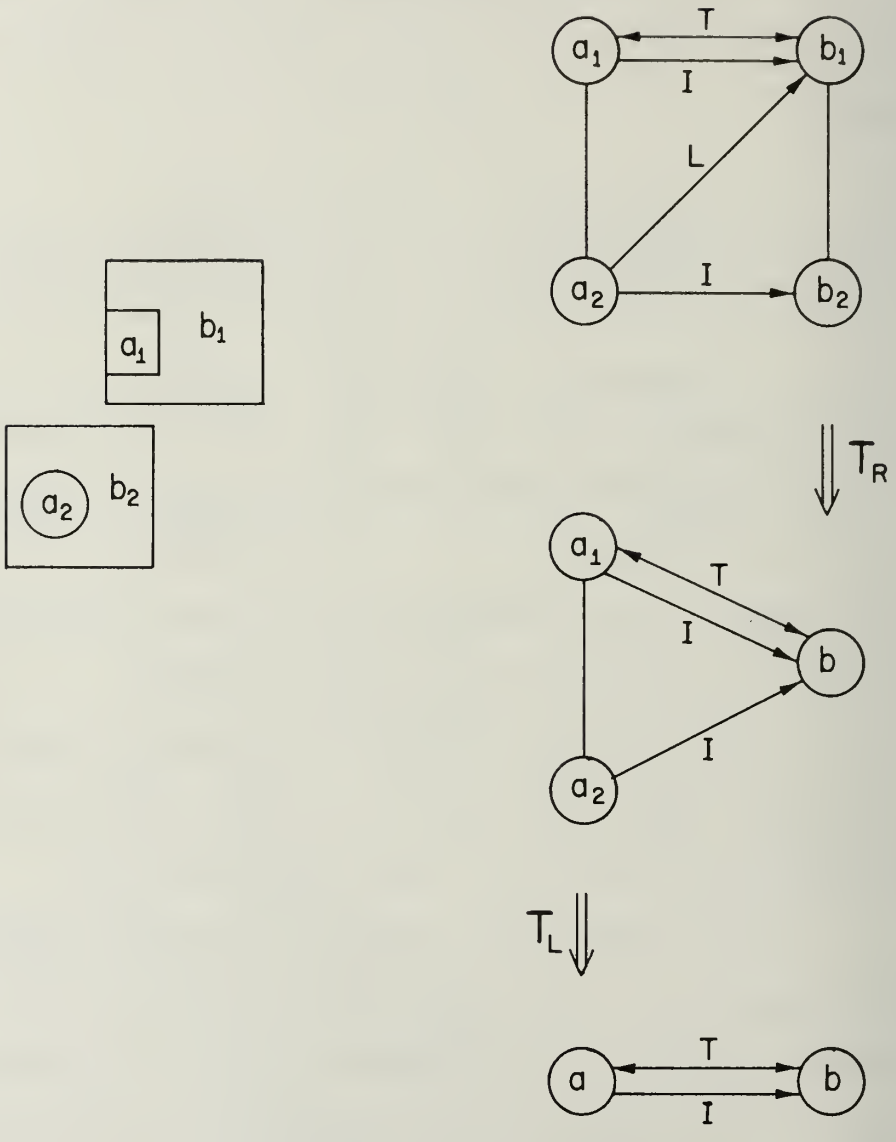


Figure 13. Composite Embedding Implied by Simple Embedding Types

Table 3 shows the simple embedding types and composite embedding type implied by the theorem for the relations used in the example.

Table 3 . Embedding Types for Picture Processing Relations

relation	description	simple embedding type	implied composite embedding type
R_T	touches	OR	ANY
R_T^{-1}	touches	OR	ANY
R_I	inside of	OR	ALL-TAILS
$R_I^{-1} = R_C$	contains	AND	ALL-HEADS
R_L	left of	AND	ALL
$R_L^{-1} = R_R$	right of	AND	ALL

The implied composite embedding types allow us to conclude that a touches b and that a is inside of b. This can be justified by applying simple embedding types to T_R and T_L. It is interesting to note that if we had applied T_L first and then T_R, the composite embedding would not have been implied by the simple types at each step. However, since $T_R \cdot T_L = T_L \cdot T_R$ the theorem is still true.

3.3 Lattice Operational Characterization of a Relation

3.3.1 Introduction

In the previous sections we considered embedding types and composite embedding types of a relation, R_u, for a transformation, T. In this section, we consider similar properties for a single relation defined between lattices, for special cases of transformations. These transformations perform lattice operations on their domains. The properties will be called lattice operational properties of the relation.

After some lattice preliminaries, systems composed of lattices M, N, and relation H are formally defined. Here we define a group of system transformations to simplify later proofs.

The lattice operational properties of systems are then defined. All possible consistent combinations of property values are established by the use of reduction theorems and by the construction of example systems. Using twelve independent properties there are exactly 136 possible combinations of property values. This set can be generated from a minimum set of thirty-three values by the group of system transformations.

3.3.2 Lattice Preliminaries

3.3.2.1 Notation

In the following, assume M and N are lattices on X and Y with the defining operations meet, $\cap$, and join, $\cup$, $M = (X, \cup, \cap)$, $N = (Y, \cup, \cap)$. The basic principles of lattices

used in this paper may be found in Chapter One of Birkhoff (91) or in Chapters One and Two of Szasz (92).

3.3.2.2 Lattice Duality

The dual of a lattice theoretical proposition involving the operational symbols $\cap$ and $\cup$ is obtained by interchanging $\cap$ and $\cup$ everywhere in the proposition. Denote the dual of a proposition P by D(P). By the lattice duality principle, P is true if and only if D(P) is true, i.e., $P \iff D(P)$.

Given a lattice $M_1 = (X, \cup_1, \cap_1)$ we can define a new lattice

$M_2 = (X, \cup_2, \cap_2)$ by

$a \cap_1 b = a \cup_2 b$

$a \cup_1 b = a \cap_2 b$ for all $a, b \in X$.

The lattice duality principle shows that M_2 is a lattice. M_2 is called the dual of M_1, $M_2 = D(M_1)$. The ordering of M_2 is reversed from that of M_1, We use less than or equal, $\leq$, and greater than or equal, $\geq$, to represent the lattice ordering.

Since $a \leq b \iff a \cup b = b$

and $a \geq b \iff a \cap b = b$

$D(a \leq b) = a \geq b$.

Clearly D(D(M))= M. If D(M) is lattice isomorphic to M then we say M is self-dual. A lattice isomorphism is a one-to-one and onto mapping which preserves both lattice operations.

We will represent finite lattices by diagrams. The upward direction in the diagram corresponds to "greater than."

If point a is higher than point b in the diagram, and if points a and b are connected by a branch, then a $\succ$ b. The diagram of the dual lattice may be obtained by inverting the diagram of the lattice.

3.3.3 Systems

3.3.3.1 Definition of a System

We consider a class of systems Ψ where $S \in \Psi$ is a system in Ψ. Let X, Y be arbitrary sets, H a binary relation from X to Y, $H \subseteq X \times Y$, and M and N be lattices on the sets X and Y respectively,

$$M = (X, \cup, \cap), \quad N = (Y, \cup, \cap).$$

Then $S \in \Psi$ if and only if $S = (H,M,N)$ for some X, Y, H, M and N.

3.3.3.2 Group of Operators on a System

Given a system $S \in \Psi$, we can derive other systems $S' \in \Psi$ by taking dual lattices and inverse relations. This will simplify the study of properties of a relation H in a system S, and the enumeration of all possible relations H with a set of properties.

Thus, we will define a group of operators, Z, which may be applied to a system S, or to a logical proposition, A, about S. If $z \in Z$ then z applied to S yields a well-defined system S', i.e. $z(S) = S'$ or $z(H,M,N) = (H',M',N')$. z applied to a logical proposition A about S yields a well-defined logical proposition A' about S'. Furthermore, A is true if and only if A' is true.

The new systems will be obtained by all possibilities of dualizing the lattices M and N, or inverting the relation H. The group G of operators can be generated by composition of a dual operator and inverse operator.

The group consists of eight operators defined in Table 4.

Table 4 . System Operators

Let $S = (H,M,N)$, $S \in \Psi$, where $H \subseteq X \times Y$, $M = (X, \cup, \cap)$. $N = (Y, \cup, \cap)$. $z(H,M,N) = (H',M',N')$, $z \in Z$, $z(S) = S', S' \in \Psi$

	T	H'	M'	N'
Identity	E	H	M	N
Dual	D	H	D(M)	D(N)
Left Dual	D_L	H	D(M)	N
Right Dual	D_R	H	M	D(N)
Inverse	I	H^{-1}	N	M
Inverse Dual	ID	H^{-1}	D(N)	D(M)
Inverse Left Dual	ID_L	H^{-1}	N	D(M)
Inverse Right Dual	ID_R	H^{-1}	D(N)	M

The group operation is composition of operators. Let $z_1, z_2, z_3 \in Z$. We denote $z_1(z_2(S))$ as $z_1 z_2(S)$, and the composition of operators z_2 and then z_1 as $z_1 z_2$.

Operation composition is associative, i.e. $z_1(z_2 z_3) = (z_1 z_2)z_3 = z_1 z_2 z_3$ which means apply z_3, z_2, z_1 in that order. The last three operators in the table are designated as compositions. For example, ID is the composition of D and I.

We display Table 5 of operator composition to show that it is a group with identity element E.

Table 5. Operator Composition

z_1 \ z_2	E	D	D_L	D_R	I	ID	ID_L	ID_R
E	E	D	D_L	D_R	I	ID	ID_L	ID_R
D	D	E	D_R	D_L	ID	I	ID_R	ID_L
D_L	D_L	D_R	E	D	ID_R	ID_L	ID	I
D_R	D_R	D_L	D	E	ID_L	ID_R	I	ID
I	I	ID	ID_L	ID_R	E	D	D_L	D_R
ID	ID	I	ID_R	ID_L	D	E	D_R	D_L
ID_L	ID_L	ID_R	I	ID	D_R	D_L	D	E
ID_R	ID_R	ID_L	ID	I	D_L	D_R	E	D

Note that all elements have order 2 except ID_L and ID_R which have order 4 and the group is not commutative, since (e.g.)

$$ID_L = D_R I \neq D_L I.$$

The group G is isomorphic to the group of symmetries of a square. This can be seen by taking I to be a reflection through the line y = x and D_L to be reflection in the Y axis.

Given a system S = (H,M,N), $S \in \Psi$, $z(S) = S'$ is a well-defined element of Ψ for all $z \in Z$. This follows from the definition of z and the fact that $H^{-1} = \{ (y,x) \mid (x,y) \in H \}$ is a well-defined binary relation between Y and X, D(M) is a well-defined lattice on X, and D(N) is a well-defined lattice on Y.

Now assume that we have a logical statement, A, about a system, S, involving the lattice operations in M and N, the relation H, variables, and expressions of logic. Then for all z in Z we can form a statement z(A) = A' about S' as shown in Table 6.

Table 6. System Operators Applied to A

z	z(A)
E	A
D_L	A with lattice operations in M replaced by their dual operations.
D_R	A with lattice operations in N replaced by their dual operations.
I	A with H replaced by H^{-1}.

The other operators in Z are compositions of the operators shown in Table 6.

For example, if

$$A = ((x, y_1 \cup y_2) \in H),$$

$$I(A) = ((x, y_1 \cup y_2) \in H^{-1}) = ((y_1 \cup y_2, x) \in H),$$

$$D_R(A) = ((x, y_1 \cap y_2) \in H)$$

$$ID_R(A) = ((x, y_1 \cap y_2) \in H^{-1}) = ((y_1 \cap y_2, x) \in H),$$

$$D_R I(A) = ((y_1 \cup y_2, x) \in H).$$

By the definition of H^{-1}, the lattice theoretical duality principle, and the fact that S' always exists and is well-defined, z changes true statements about S = (H, M, N) to true statements about S' = (H', M', N'). Thus, a logical statement concerning S is true if and only if the transformed statement A' = z(A) concerning S' = z(S) is true. That is, $A \Leftrightarrow z(A)$.

3.3.4 Types of Transformations Considered

The transformations considered in the definition of the lattice operational properties of a relation, H, are of the three basic types given below.

$$T_1 : b_1 \rightarrow b \quad \text{where } b = T(b_1) \geq b_1 \quad \text{or } b = T(b_1) \leq b_1$$

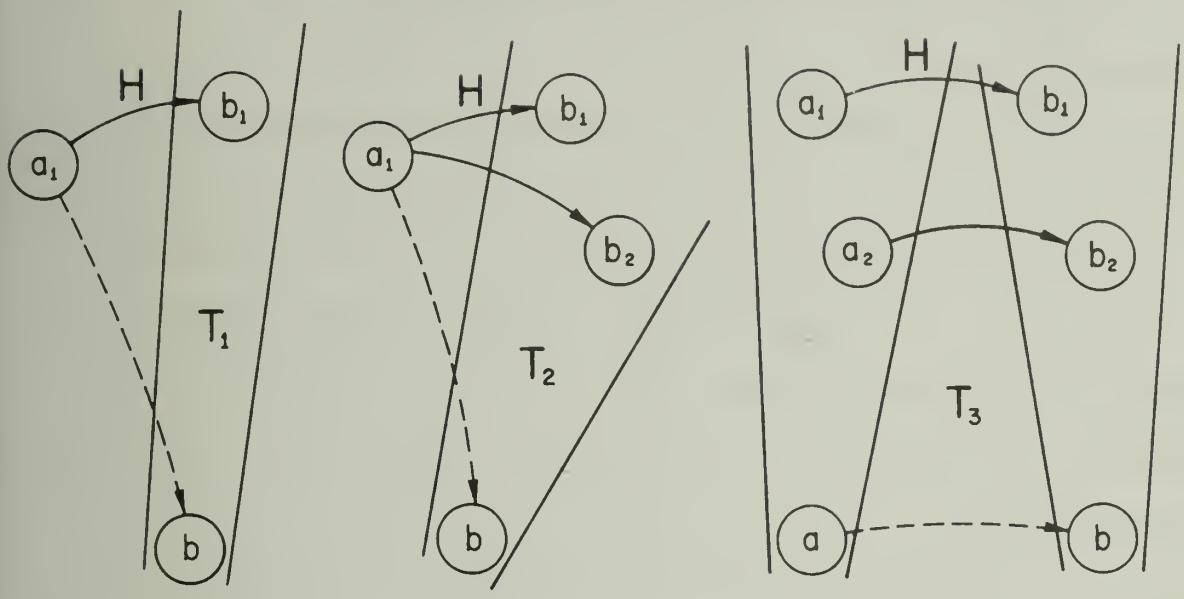
$$T_2 : \{b_1, b_2\} \rightarrow b \quad \text{where } b = T\{b_1, b_2\} = b_1 \cup b_2 \quad \text{or} \quad = b_1 \cap b_2$$

$$T_3 : \{a_1, a_2\} \rightarrow a \quad \text{and} \quad \{b_1, b_2\} \rightarrow b$$

$$\text{where } a = T\{a_1, a_2\} = a_1 \cup a_2 \text{ or } a_1 \cap a_2$$

$$\text{and } b = T\{b_1, b_2\} = b_1 \cup b_2 \text{ or } b_1 \cap b_2$$

The basic transformations as well as the context considered for the properties of H are illustrated in Figure 14 .



P1: $b \geq b_1$ P3: $b_1 \cup b_2$ P5: $a_1 \vee a_2$ $b_1 \vee b_2$

P2: $b \leq b_1$ P4: $b_1 \wedge b_2$ P6: $a_1 \cup a_2$ $b_1 \wedge b_2$

P7: $a_1 \wedge a_2$ $b_1 \wedge b_2$

$P6^{-1}$: $a_1 \wedge a_2$ $b_1 \vee b_2$

Figure 14 . Basic Operational Transformations

3.3.5 Definition of Properties

We will now define properties of a relation H in a system $S \in \Psi$, S = (H, M, N). The properties considered involve the preservation of the lattice operations : meet, join, less than or equal, and greater than or equal, in one or both lattices.

A property P is a logical statement about S and thus

$z(P)$, $z \in Z$, is a statement about $z(S)$. We will define twelve properties which are closed under operations in Z. The properties are logically independent and imply all lattice operational properties of a system S. The properties will be denoted: P1, $P1^{-1}$, P2, $P2^{-1}$, P3, $P3^{-1}$, P4, $P4^{-1}$, P5, P6, P7, $P6^{-1}$ In some cases the P will be omitted. I(P) is denoted by P^{-1} unless I(P) = P in which case P is a symmetric property. P5 and P7 are symmetric properties. The properties are defined below, where a's and b's are used as variables in M and N, respectively.

Recall, according to the definition, that I(P) is formed by replacing H by H^{-1} in P. For consistency in the definition of P^{-1}, we also interchange a and b variables. This is only a notational convenience so that a's and b's still refer to variables in M and N, respectively, since H goes between M and N in S. For example, the definition of $P3^{-1}$ is obtained from the definition of P3 as follows:

$$P3: aHb_1 \;\&\; aHb_2 \Rightarrow aH(b_1 \cup b_2)$$

$$I(P3) = P3^{-1}: aH^{-1}b_1 \;\&\; aH^{-1}b_2 \Rightarrow aH^{-1}(b_1 \cup b_2) \Leftrightarrow$$

$$b_1Ha \;\&\; b_2Ha \Rightarrow (b_1 \cup b_2)\,Ha$$

changing notation:

$$P3^{-1}: a_1Hb \;\&\; a_2Hb \Rightarrow (a_1 \cup a_2)Hb$$

Definition of Properties:

P1: Propagates Images Up

$$aHb_1 \;\&\; b_2 \geq b_1 \Rightarrow aHb_2$$

$P1^{-1}$: Propagates Inverse Images Up

$a_1Hb \;\&\; a_2 \geq a_1 \Rightarrow a_2Hb$

P2: Propagates Images Down

$aHb_1 \;\&\; b_2 \leq b_1 \Rightarrow aHb_2$

$P2^{-1}$: Propagates Inverse Images Down

$a_1Hb \;\&\; a_2 \leq a_1 \Rightarrow a_2Hb$

P3: Joins Images

$aHb_1 \;\&\; aHb_2 \Rightarrow aH(b_1 \cup b_2)$

$P3^{-1}$: Joins Inverse Images

$a_1Hb \;\&\; a_2Hb \Rightarrow (a_1 \cup a_2)Hb$

P4: Meets Images

$aHb_1 \;\&\; aHb_2 \Rightarrow aH(b_1 \cap b_2)$

$P4^{-1}$: Meets Inverse Images

$a_1Hb \;\&\; a_2Hb \Rightarrow (a_1 \cap a_2)Hb$

P5: Preserves Joins

$a_1Hb_1 \;\&\; a_2Hb_2 \Rightarrow (a_1 \cup a_2)H(b_1 \cup b_2)$

P6: Dualizes Joins

$a_1Hb_1 \;\&\; a_2Hb_2 \Rightarrow (a_1 \cup a_2)H(b_1 \cap b_2)$

$P6^{-1}$: Dualizes Meets

$a_1Hb_1 \;\&\; a_2Hb_2 \Rightarrow (a_1 \cap a_2)H(b_1 \cup b_2)$

p7: Preserves Meets

$$a_1 H b_1 \ \& \ a_2 H b_2 \Rightarrow (a_1 \wedge a_2) H (b_1 \wedge b_2)$$

Other properties can be defined which are logical combinations of the twelve independent properties. We show four such properties below.

$P2 \ \& \ P2^{-1}$: Preserves Less Than

$P1 \ \& \ P1^{-1}$: Preserves Greater Than

$P2^{-1} \ \& \ P1$: Dualizes Less Than

$P1^{-1} \ \& \ P2$: Dualizes Greater Than

Table 7 shows z(P) for all twelve properties P and all z in Z. Since each z(P) is a permutation of the properties P, we can determine the true or false values of the properties z(P) of the system z(S), given the properties P of S.

Table 7 . System Operators Applied to Properties

z \ P	z(P)											
	1	1^{-1}	2	2^{-1}	3	3^{-1}	4	4^{-1}	5	6	7	6^{-1}
D	2	2^{-1}	1	1^{-1}	4	4^{-1}	3	3^{-1}	7	6^{-1}	5	6
D_L	1	2^{-1}	2	1^{-1}	3	4^{-1}	4	3^{-1}	6^{-1}	7	6	5
D_R	2	1^{-1}	1	2^{-1}	4	3^{-1}	3	4^{-1}	6	5	6^{-1}	7
I	1^{-1}	1	2^{-1}	2	3^{-1}	3	4^{-1}	4	5	6^{-1}	7	6
ID	2^{-1}	2	1^{-1}	1	4^{-1}	4	3^{-1}	3	7	6	5	6^{-1}
ID_L	1^{-1}	2	2^{-1}	1	3^{-1}	4	4^{-1}	3	6	7	6^{-1}	5
ID_R	2^{-1}	1	1^{-1}	2	4^{-1}	3	3^{-1}	4	6^{-1}	5	6	7

Table 7 also shows that the twelve properties are partitioned into three sets of four properties such that each set is closed under all z in Z. These three sets are separated by double lines in Table 7 , and will be referred to in the following as the P1 set, the P3 set, and the P5 set. Any combination of values of the twelve properties will be referred to by an index, (i, j, k), where i, j, k are indices of the values of the properties in the P1, P3, and P5 set respectively. The values of the indices i, j, k are determined by partitioning the sixteen possible true or false values of four properties into the following six sets, with indices 0, 1, 2, ..., 5:

Index	Set
0	0000
1	0001, 0010, 0100, 1000
2	1010, 0101
3	0011, 0110, 1001, 1100
4	1110, 1101, 1011, 0111
5	1111

Each of these six sets of property values is closed under all z in Z, and each element of each set is a generator of the set under z in Z.

Thus the index, (i, j, k) of any value combination of the twelve properties does not change under z in Z. That is, if values P are indexed by (i, j, k), the values z(P) are also indexed by (i, j, k). These facts will allow us to easily derive a minimum set of generators for all consistent cases of properties. This derivation will be carried out after proving theorems interrelating the twelve properties in the next section.

3.3.6 Enumeration of Possible Combinations of Properties

3.3.6.1 Preliminary Remarks

We now wish to determine all logically consistent combinations of the twelve properties P of a relation H in a system S = (H, M, N).

This section reduces the number of possibly consistent value combinations by proving and applying theorems which interrelate the twelve properties.

In Schwebel and McCormick (70), the proof that this set of theorems is complete is given by constructing examples of systems having all value combinations in the reduced set.

We will prove theorems involving properties P and expressions of logic. Corresponding to a theorem, A, are eight theorems z(A), $z \in Z$, which may or may not be independent of A, which are obtained by replacing P by z(P) for all P involved in A. Since $z(P) \Leftrightarrow P$, and the properties involve arbitrary system S, it follows directly that $z(A) \Leftrightarrow A$. That is, a proof of A implies a proof of z(A). This will considerably simplify the proofs to establish all consistent cases.

3.3.6.2 Theorems Relating Properties

In Table 8 we list twenty independent theorems, numbered one through twenty, and a diagrammatic representation of the theorems in a form which will make the application of the theorems in the reduction process readily apparent.

Table 8 . Theorems

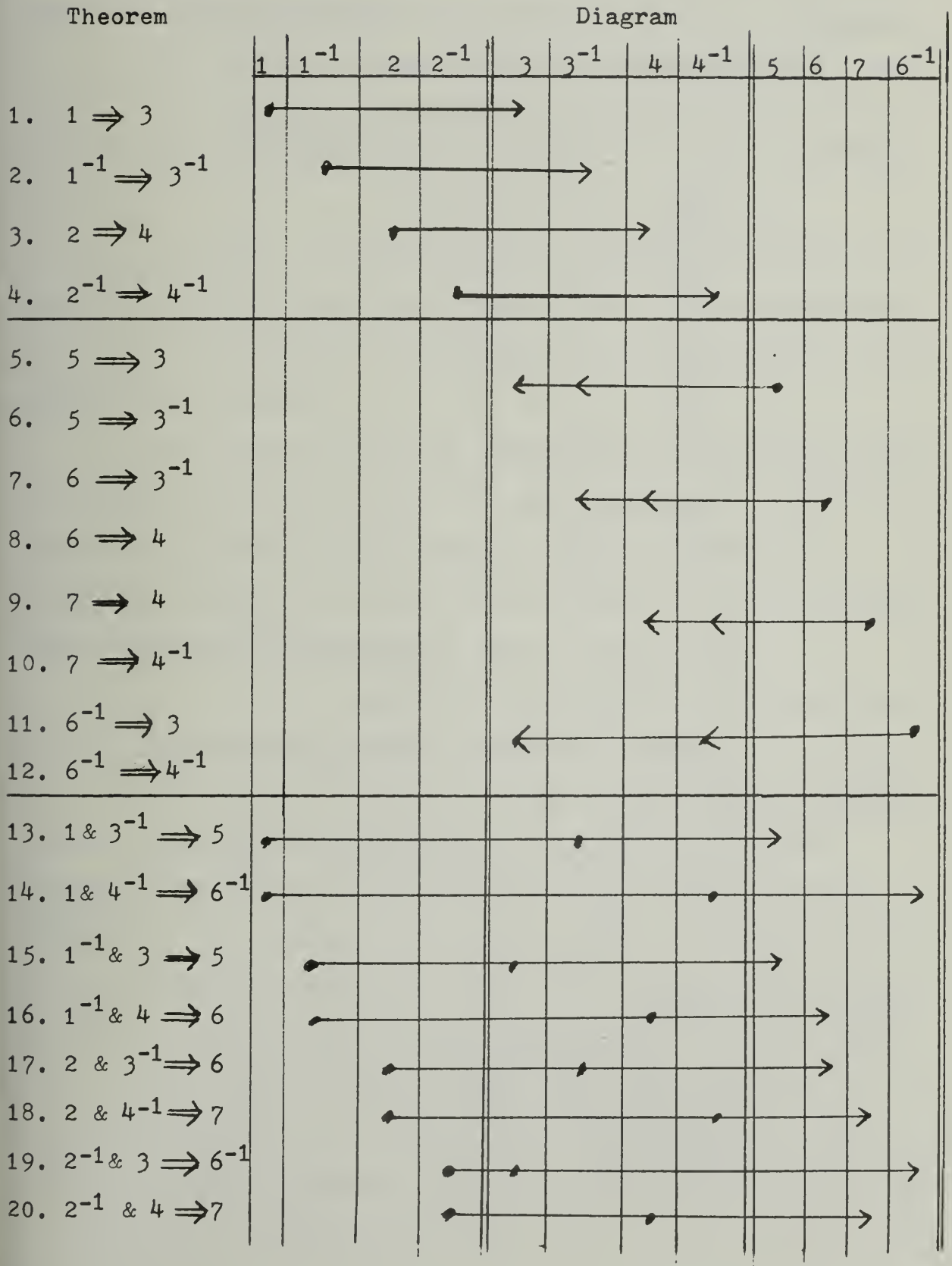


Table 9 shows that transformations, z, applied to Theorems 1, 5, and 13 generate all twenty theorems, and that these twenty theorems are closed under z in Z.

Table 9. System Operators Applied to Theorems

z \ A	z(A) 1	5	13
D_L	1	11	14
I	2	6	15
ID_L	2	7	16
D_R	3	8	17
D	3	9	18
ID_R	4	12	19
ID	4	10	20

Below we give proofs of theorems 1, 5, and 13, thus implying proofs of all twenty theorems.

Thm. 1 $P1 \Rightarrow P3$

Proof: aHb_1 & aHb_2 $\Rightarrow$

(by P1 and $b_1 \vee b_2 \geq b_1$ or $b_1 \vee b_2 \geq b_2$)

$aH(b_1 \vee b_2)$ Q.E.D.

Thm. 5 $P5 \Rightarrow P3$

Proof: aHb_1 & aHb_2 $\Rightarrow$ (by P5)

$(a \vee a)H(b_1 \vee b_2)$ $\Rightarrow$

$aH(b_1 \vee b_2)$ Q.E.D.

Thm. 13 $P1 \;\&\; P3^{-1} \Rightarrow P5$

Proof: $a_1 H b_1 \;\&\; a_2 H b_2 \Rightarrow$

(by $(b_1 \cup b_2) \geq b_1$ and $(b_1 \cup b_2) \geq b_2$ and P1)

$a_1 H (b_1 \cup b_2) \;\&\; a_2 H (b_1 \cup b_2) \Rightarrow$ (by $P3^{-1}$)

$(a_1 \cup a_2) H (b_1 \cup b_2)$ Q.E.D.

Figure 15 represents all of the twenty theorems by an implication graph whose nodes represent the twelve properties. The following symmetries of the properties and implications are apparent from Figure 15 :

The transformation D_R corresponds to reflection about a line through nodes 1^{-1} and 2^{-1}. D_L corresponds to reflection about a line through nodes 3 and 4. D(P) corresponds to reflection through the center of the diagram on a line through node P, for any property P.

3.3.6.3 Reduction of Possible Consistent Cases

There are $2^{12} = 4096$ possible combinations of binary values of the twelve properties. By applying the twenty theorems, the number of possible consistent combinations is reduced to 136. Further, the set of values, V, where $V = \{v\}$, is partitioned into subsets, called blocks, by the equivalence relation: v is equivalent to v' if and only if $v = z(v')$ for some z in Z. Every element in each block is a generator of the block under z in Z. We will show that there are thirty-three consistent blocks and, therefore, thirty-three generators can generate all consistent cases.

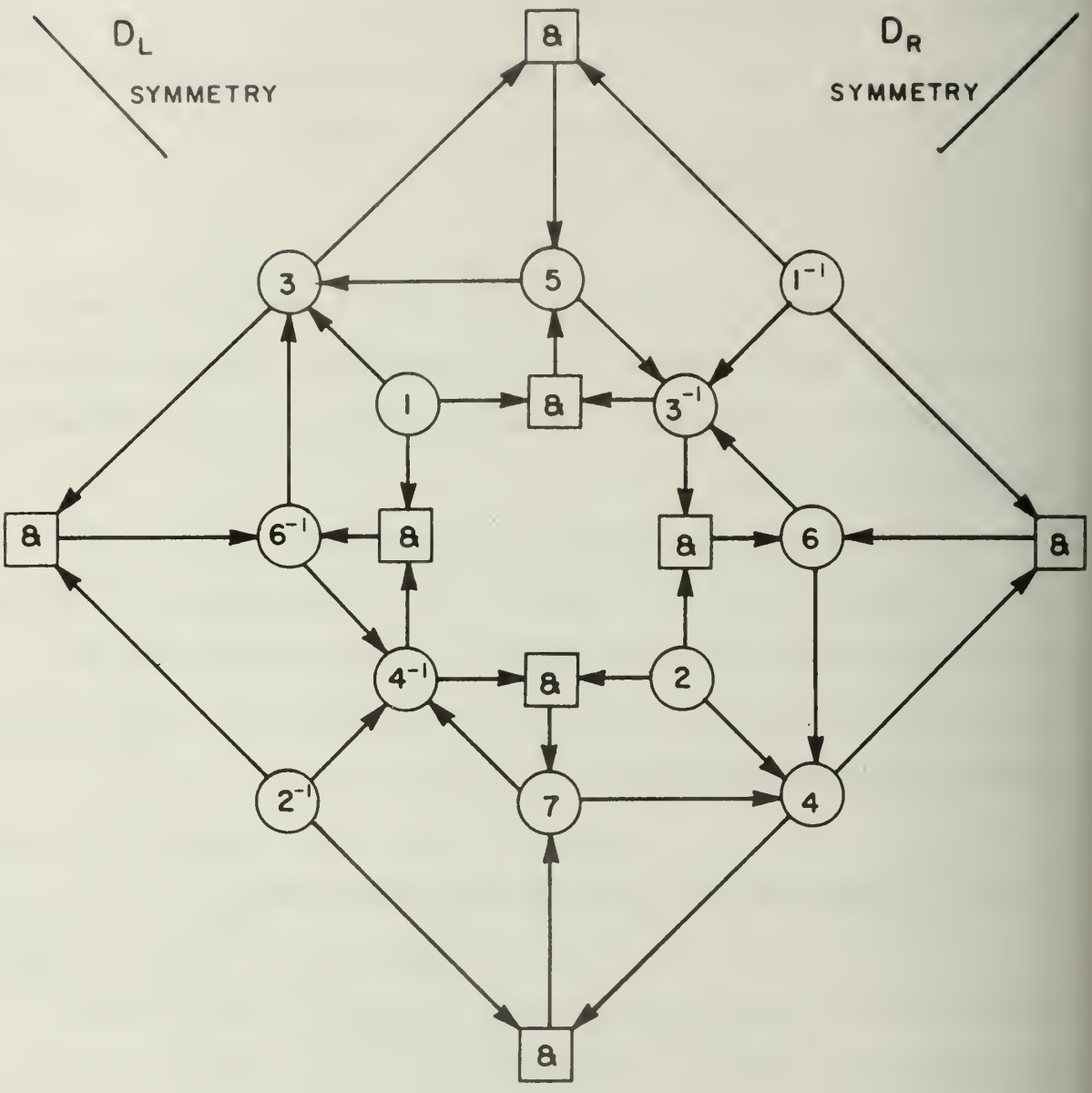


Figure 15. Implication Graph for Twelve Properties

The reduction process and derivation of a minimum set of generators is illustrated in Table 10 , Each combination is indexed by (i, j, k) as explained in the previous section. We will refer to a value, v, by $v_1v_2v_3$, where v_1 is the 4 bit value of the P1 set of properties, v_2 of the P3 set and v_3 of the P5 set. In the derivation, for each j = 0, 1, 2, 3, 4 all possible consistent values of v_1 and of v_3 are listed for one possible value of v_2. For j = 5, since v_2 has only one value, all possible consistent values of v_3 are listed for one value of v_1 for each i = 1, 2, 3, 4. For i = 0, j = 5, one value of v_3 is listed for each k = 0, 1, 2, 3, 4, 5.

It is easily seen that the values in Table 10 are generators for all possible consistent values. For any consistent value v_1 v_2 v_3, with index (i, j, k), we can always find a z such that $z(v_1\ v_2\ v_3) = v_1', v_2', v_3'$, and v_1', v_2', v_3' is in Table 10 . There are three cases:

1) j = 0, 1, 2, 3, 4 and z is the transformation which takes v_2 into the v_2' used in the table.

2) j = 5, i = 1, 2, 3, 4 and z takes v_1 into the v_1' used in the table.

3) j = 5, i = 0 and z takes v_3 into the v_3' used in the table.

Since there are thirty-two different (i, j, k) indices in Table 10 , and z does not alter the index of its operand, there are at least thirty-two consistent blocks.

Table 10a. Reduction of Set of Possible Consistent Values of Twelve Properties

All possible values in all i, k sets are listed for one representative value in the j set, for j = 0,1,2,3,4.

Index	z	Properties		
i j k		$1\ 1^{-1}\ 2\ 2^{-1}$	$3\ 3^{-1}\ 4\ 4^{-1}$	$5\ 6\ 7\ 6^{-1}$
000		0000	0000	0000
010		0000	0001	0000
110		0001		0000
020		0000	1010	0000
120		0010		0000
120	D	1000		0000
220		1010		0000
030		0000	0011	0000
031		0000		0010
131		0001		0010
131	I	0010		0010
331		0011		0010
040		0000	1110	0000
041		0000		0100
041	D_R	0000		1000
043		0000		1100
141		0010		0100
143(1)		0010		1100
143(2)		0100		1100
343		0110		1100
141	D_R	1000		1000
143(1)	D_R	1000		1100
243		1010		1100
343	D_R	1100		1100
443		1110		1100

Table 10b. Reduction of Set of Possible Consistent Values of Twelve Properties

All possible values for all k sets are listed for j = 5 and for one representative value in the i set, for i = 1, 2, 3, 4, 5. For i = 0, j = 5 one representative value is listed for each k = 0, 1, 2, 3, 4, 5.

Index	z	Properties		
i j k		$1\ 1^{-1}\ 2\ 2^{-1}$	$3\ 3^{-1}\ 4\ 4^{-1}$	$5\ 6\ 7\ 6^{-1}$
050		0000	1111	0000
051				0001
052				1010
053				0011
054				1110
055				1111
153		0001	1111	0011
154				0111
154	D_R			1011
155				1111
255		1010	1111	1111
354		0011	1111	0111
355				1111
455		1110	1111	1111
555		1111	1111	1111

Table 11 Minimum Set of 33 Generators of 136 Cases

Block Index i j k	$1\ 1^{-1}\ 2\ 2^{-1}$	$3\ 3^{-1}\ 4\ 4^{-1}$	$5\ 6\ 7\ 6^{-1}$	Order of Block
000	0000	0000	0000	1
010	0000	0001	0000	4
020	0000	1010	0000	2
030	0000	0011	0000	4
031	0000	0011	0010	4
040	0000	1110	0000	4
041	0000	1110	0100	8
043	0000	1110	1100	4
050	0000	1111	0000	1
051	0000	1111	0001	4
052	0000	1111	1010	2
053	0000	1111	0011	4
054	0000	1111	1110	4
055	0000	1111	1111	1
110	0001	0001	0000	4
120	0010	1010	0000	4
131	0001	0011	0010	8
141	0010	1110	0100	8
143(1)	0010	1110	1100	8
143(2)	0100	1110	1100	4
153	0001	1111	0011	4
154	0001	1111	0111	8
155	0001	1111	1111	4
220	1010	1010	0000	2
243	1010	1110	1100	4
255	1010	1111	1111	2
331	0011	0011	0010	4
343	0110	1110	1100	8
354	0011	1111	0111	4
355	0011	1111	1111	4
443	1110	1110	1100	4
455	1110	1111	1111	4
555	1111	1111	1111	1

By applying transformations to the values in Table 10 , we see that there are at most thirty-three consistent blocks. The set indexed by (1, 4, 3) requires two generators, while the other consistent (i, j, k) sets require only one generator. We index the thirty-three possibly consistent blocks by (i, j, k) if (i, j, k) $\neq$ (1, 4, 3), and by (1, 4, 3) (1) and (1, 4, 3) (2).

Table 11 lists a set of thirty-three generators from the values of Table 10 . The values in Table 10 which are not used in Table 11 have an entry under column z specifying the transformation which generates them from the value in Table 11 in the same block. Table 11 also lists the number of elements in each block. All 136 values are listed in Schwebel and McCormick (70).

3.4 Combined Properties

3.4.1 Embedding Types and Lattice Operational Properties

Since lattice operational properties are special types of embedding rules, they are implied by the simple embedding types and the composite embedding types defined earlier. Table 12 lists these implications.

Table 12 . Implications Between Embedding Types and Lattice Operational Properties

IF:	THEN it has the properties marked		
H has embedding type:	$11^{-1}22^{-1}$	$33^{-1}44^{-1}$	5676^{-1}
OR	1x1x	1x1x	xxxx
AND	x1x1	x1x1	xxxx
H^{-1} has embedding type:			
OR	x1x1	x1x1	xxxx
AND	x1x1	x1x1	xxxx
H has composite embedding type:			
ANY	1111	1111	1111
ALL-TAILS	1111	1111	1111
ALL-HEADS	1111	1111	1111
ONE-ALL	1111	1111	xxxx
ALL-ONE	1111	1111	xxxx
ALL	1111	1111	xxxx

Table 12 is easily derived from the transformation, T_1, T_2, or T_3, used to define the properties and the relations assumed on the domains. The lattice operational properties are not used. That is, referring to the notation of Figure 14 , the relationships between b and $\{b_i\}$ are not used, but only the context of the transformations shown in Figure 14 .

Thus, we could define more general properties corresponding to the three transformations in Figure 14 and have implications corresponding to those shown in Table 12

For the case of lattice operational properties, we

can determine the logically consistent combinations of properties which can be implied from Table 12 by examining the consistent cases given in Table 12. This results in seventeen cases which are allowed assuming the embedding type OR or AND for H or H^{-1}: namely, the cases generated by the 220, 243, 255, 443, 455, and 555 blocks. Only one case is allowed assuming a composite embedding type for H: namely, the case in the 555 block.

The fact that only a small proportion, 17/136, of the cases of consistent properties are implied by the more general relation embedding types partially justifies the characterization of lattice operational properties. If these properties are important in a graph-transformational system, then the study of the structure of the sets is necessary.

3.4.2 Lattice Operation Root Transformations

Suppose the root operations for composite-forming graph transformations are lattice operations. Then we want to investigate the relationship between the root transformations defined in Chapter Two and the lattice operational properties.

We thus assume that the root binary operation forms the meet or join of two elements, $\{a, (a, A, b), b\} \rightarrow a \cup b$, e.g., RT_1: $\{a, (a, A, b), b\} \rightarrow a \cup b$. Then the root transformations RQ2, RQ3, and RQ4 become statements involving lattice operations. We will denote these transformations

LRQ2, LRQ3, and LRQ4. They are illustrated in Figure 16.

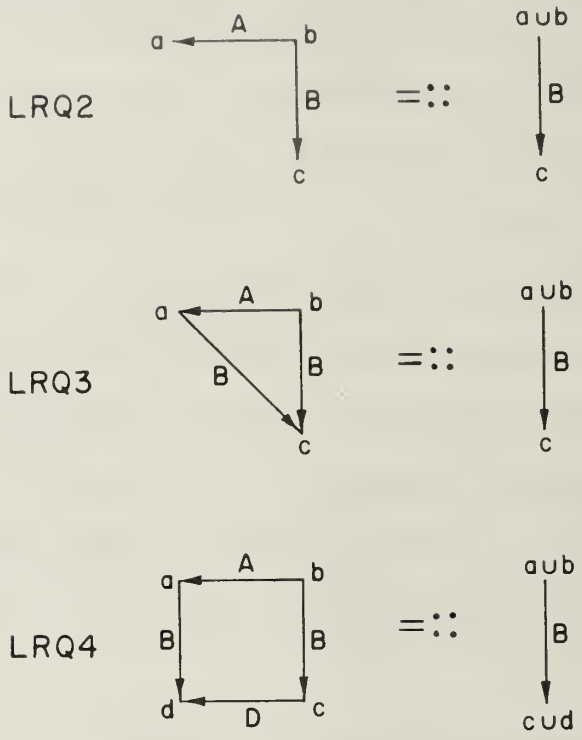


Figure 16. Lattice Root Transformations

Now we can compare the root transformations with the lattice operational properties. Note that in some cases they are directly comparable, for instance: LRQ3 and $P3^{-1}$ are related by:

$P3^{-1}$ ⇒ LRQ3(*, H, a_1, a_2, b) where a_1, $a_2 \in M$ and $b \in N$. That is, if $P3^{-1}$ is true then LRQ3(A, H, a_1, a_2, b) is true.

regardless of the branch (a_1, A, a_2). Thus, the root lattice transformations have an additional constraint relation beyond their corresponding lattice operational properties. Similarly, the other properties on three elements are comparable to dual and inverse operators of the root transformation LRQ3; the other properties on four elements are comparable to dual and inverse operators on LRQ4. These relationships are shown in Table 13 .

Using the operators, z, in Z we can obtain root transformations involving the dual and inverse operators:

$D, D_L, D_R, I, ID, ID_L, ID_R$.

Table 13. Lattice Operational Properties Compared to Lattice Operation Root Transformations

Lattice operational property	Equivalent transformation
	Variable position (A, B, a, b, c)
$P3 = I\ (P3^{-1})$	$I\ (LRQ3(*, H, a_1, a_2, b))$
$P3^{-1}$	$LRQ3(*, H, a_1, a_2, b)$
$P4 = ID\ (P3^{-1})$	$ID(LRQ3(*, H, a_1, a_2, b))$
$P4^{-1} = D\ (P3^{-1})$	$D\ (LRQ3(*, H, a_1, a_2, b))$
	(A, B, D, a, b, d, c)
P5	$LRQ4(*, H, *, a_1, a_2, b_1, b_2))$
$P6 = D_R\ (P5)$	$D_R(LRQ4(*, H, *, a_1, a_2, b_1, b_2))$
$P7 = D\ (P5)$	$D\ (LRQ4(*, H, *, a_1, a_2, b_1, b_2))$
$P6^{-1} = D_L(P5)$	$D_L(LRQ4(*, H, *, a_1, a_2, b_1, b_2))$

* = don't care

This concludes our study of the embedding types and lattice operational properties of relations involved in transformations. In the next chapter we examine other considerations important for the application of graph transformations.

4. APPLICATION CONSIDERATIONS

4.1 Preliminary Remarks

The basic problem of this work is to develop a methodology for the description of pictures. The means of obtaining these descriptions is by parsing, i.e applying graph-structure transformations selected from within a predefined production system.

We have exhibited certain necessary properties of the transformations for a parse to be well-defined and possible. We have also shown these properties for logically simple transformations. We have preferred to employ both logically simple and information-lossless transformations.

Now we will consider briefly how the graph-structure transformational framework meshes with the picture processing schema mentioned in Chapter One. The crucial question here is the choice of the domain for the transformation. In this connection interaction with unstructured classification techniques is required. We are also interested in the way that model information is represented in the graph-transformational system, in contrast to model-directed systems. In addition, the criteria which may be used to judge the optimality of transformations or of complete parses are of great interest.

4.2 Domain Choice for Application of Transformations

In order to pick candidate graphs for transformations we have emphasized logical criteria such as the type of domain graph and the simplicity of application.

Unstructured classification techniques on associated attributes provide a first and major reduction of the candidate graphs. That is, attribute clustering techniques are applied first so that only graphs with valid ranges or intervals of attribute values remain as candidates. For example, if color is available as an attribute in the search for abnormal cells in a white blood cell differential, a large percentage of candidates are eliminated after attribute analysis is applied.

4.3 Model Aspects and Optimality

To determine if a parse is legal and optimal, information about the class of objects being described is necessary.

One approach is to have a model or set of models of known objects. If the models are expressed by graph structures, then a parse would attempt to proceed from the initial graph structure to a model graph structure. Overall, this class of techniques can be called graph-matching techniques. Their main disadvantage is inefficiency due to the many comparisons which must be attempted. It is also necessary to build flexibility into the models to allow recognition of a class of

objects larger than the number of models. Of course, the techniques used to increase flexibility can also improve the efficiency of searching.

A model-directed parsing scheme may be inefficient because too much information is present in the model, i.e. in the description. The key point is that the simplest description has only that information which is necessary to distinguish the object in question from objects which are not in the class being described but still likely to be in the immediate environment.

The graph-structure parsing system need use no explicit models. The parse attempts to achieve valid and logically simple descriptions of classes of objects. An optimal parse achieves the simplest description of an object necessary to distinguish it from objects not in its class. Information about a class of objects can best be provided by giving examples of objects in the class and objects not in the class. The class information is used to select allowable graph transformations. Starting from an initial graph structure, transformations make simplifying changes in the description. The most important information is obtained from examples whose descriptions are logically close to, but outside of, the set of descriptions of objects valid to the class. These examples are called "near-miss" examples as in Winston (64). An example of a house picture, from Winston, and a near-miss to the house is shown in Figure 17 of the next

section. Transformations which modify the structure so that it matches a near-miss structure are prohibited. This is analogous to the motive of the interval covering theory of Michalski and McCormick (47) and Michalski (48): descriptions are simplified by obtaining the lowest-cost cover of a class of objects. A simplification of a description usually increases the number of objects which it covers.

In our case, the description (graph structure) is modified by transformations which expand the space of described objects without including any known near-misses.

Transformations are selected which are simplifying and yet do not transform to near-miss structures; i.e. the allowable information loss is constrained by near-miss objects.

4.4 Example Application

To illustrate the application of graph-structure transformations, we will reformulate the structural description learning system of Winston (64) using graph transformations to alter models and to construct a description.

Winston's system involves:

- .obtaining networks to describe example scenes
- .using matching techniques to link nodes in two networks
- .obtaining a skeleton network common to the matched networks
- .comparing matched nodes by types of "comparison-notes"

> .transforming the skeleton according to the type of comparison-notes to build up a model

A comparison-note compares structure in a model to structure in a near-miss or in a valid example. It represents the difference between the model and the instance presented. We will formulate a comparison note as a graph, which is then the domain of a transformation which infers a new model.

Our work aims to simplify the representation of structural and relational composites. The concepts which are fundamental to simplifying descriptions are also fundamental to learning descriptions.

Winston has stated (64, p. 244):

> ... more knowledge about the priorities of differences should lead to far better programs that do not use numbers at all...(to score differences between models and examples).

The different types of comparison-notes and resulting model changes used by Winston can be expressed by a smaller number of graph-structure transformations. The priority of a difference can then be expressed by the degree of reversibility of the transformation.

A basic transformation forms a new model from the current model and a valid example of a near-miss. In Figure 17 a graph model for house which is linked to a graph model of a near miss is transformed into a new model for house. Here S = "has subpart," R = "supported by," W = wedge and B = brick. L links between matched nodes. If the near-miss had been a valid example instead, then the model for

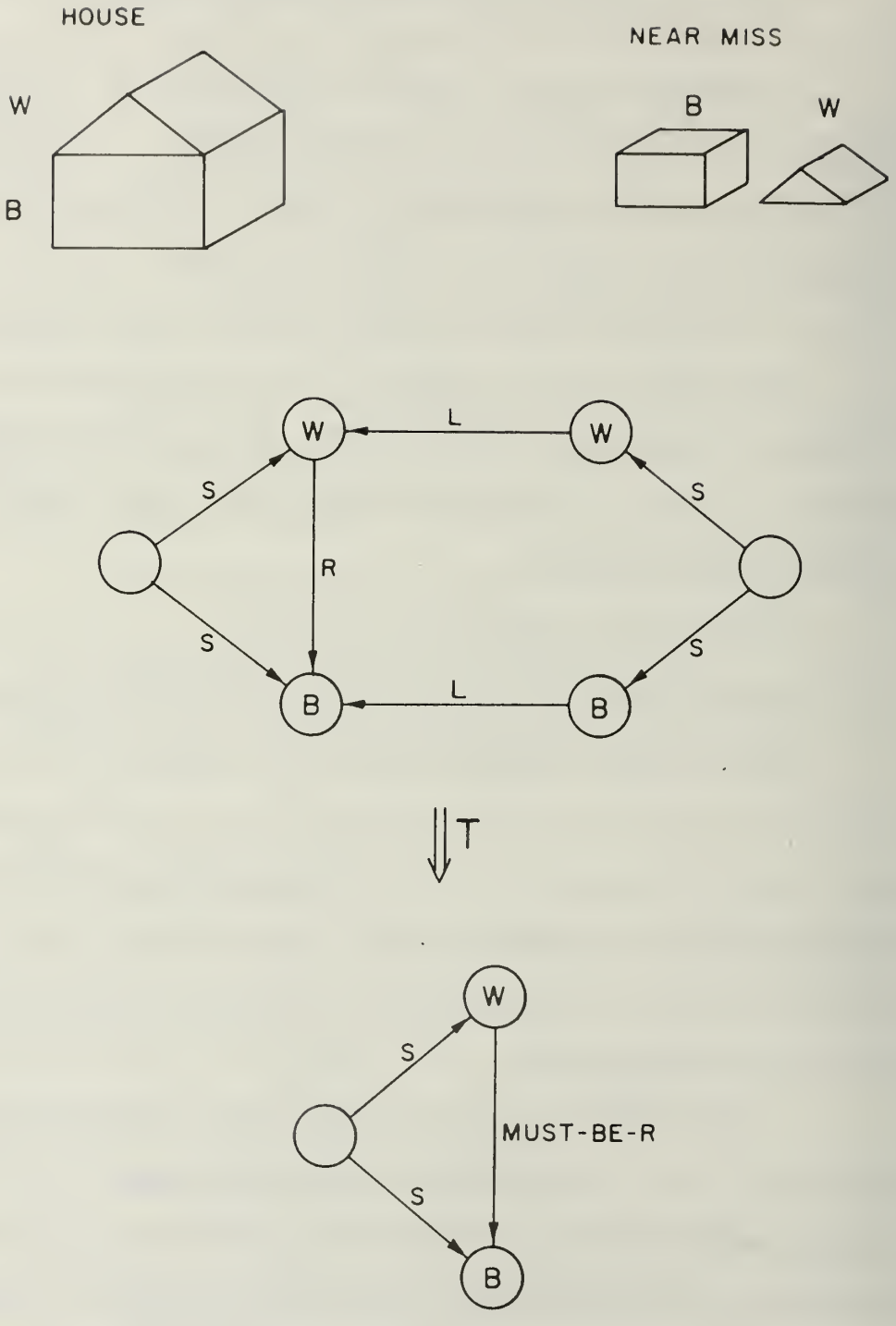


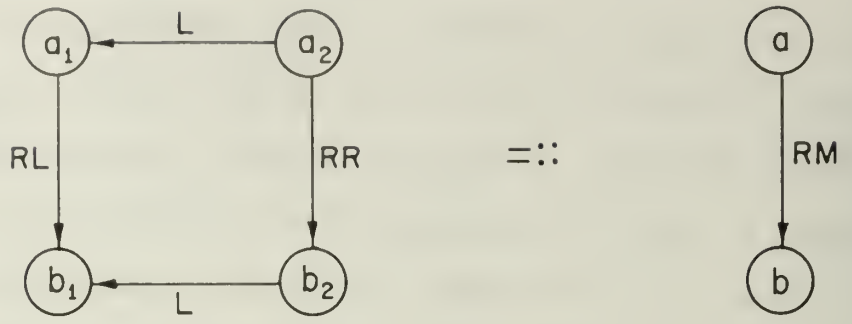
Figure 17. A Transformation Which Infers a New Model from a Valid Example and a Near-Miss

house could have been simplified by deleting the relation R.

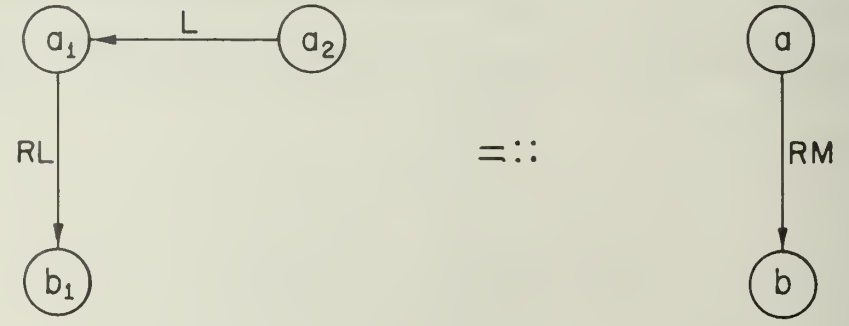
We formulate this process in terms of a small number of graph transformations. Using properties of the binary relations and duality of the spaces of the transformations reduces the size of the set of basic transformations. In this way, a simplifying description clarifies the problem.

Figure 18 illustrates the basic transformations, used to infer a new model graph. Each case of the variables for relations or nodes, whose values are listed below the diagram, corresponds to a different graph (derived from a comparison-note of Winston) and associated transformation. In the example of Figure 17, the transformation T corresponds to the first transformation in Figure 18a, with RL = R, RR = not-R, and L = near-miss-to.

In summary, this application attempts to derive a description which is logically optimal by generalizing an initial description, by means of graph transformations.

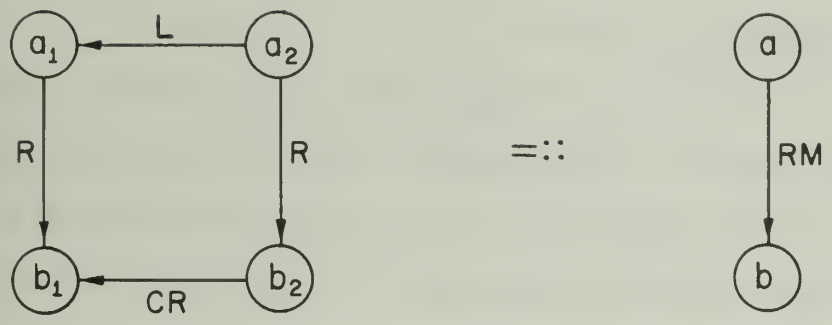


RL	RR	L	RM
R	not-R	example-of	∅
R	not-R	near-miss-to	must-be-R
R	must-be-R	example-of	R
R	must-not-be-R	example-of	contradiction

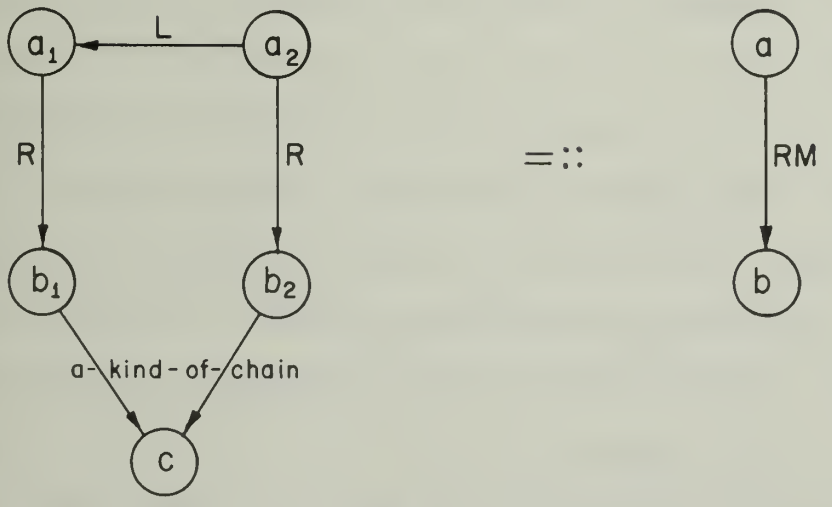


RL	L	RM
R	example-of	∅
R	near-miss-to	must-be-R
not-R	example-of	∅
not-R	near-miss-to^{-1}	must-not-be-R
R	example-of^{-1}	∅
R	near-miss-to^{-1}	must-not-be-R
not-R	example of^{-1}	∅
not-R	near-miss-to^{-1}	must-be-R
must-be-R	example-of	contradiction

Figure 18a. Basic Transformations on Compared Models



CR	L	RM	b
a-kind-of-chain	example-of	R	b_1
a-kind-of-chain	near-miss-to	must-not-be-R	b_2
a-kind-of^{-1}	near-miss-to	must-be-R	b_1



L	RM	b
example-of	R	c
example-of	∅	
near-miss-to	must-be-R	b_1
near-miss-to	must-not-be-R	b_2

Figure 18b. Basic Transformations on Compared Models

5. STRUCTURE OPERATION LANGUAGE

5.1 Introduction

The Structure Operation Language, SOL, is a computer language designed to implement graph-structure representations and necessary operations for performing structure transformations. SOL can thus be used as a picture processing language operating on graph-structured picture representations. Numerous other applications for SOL exist in areas where a graph or graph-structured representation is used.

SOL may be described as a graph-structure processing language. It does not fall into the class of linguistic or syntactic-oriented picture processing models or descriptive graphic languages, which were included in the survey of Chapter 1.

In this work, we have no explicit picture description language for describing pictures independently of processing algorithms. Most current picture description languages do imply an associated processing method. It seems realistic to first define the structure requirements which are forced by processing algorithms, and not to fix (by description languages and recognition systems) a complete picture processing model.

The initial design of the language will be done without an implementation-restricted memory structure or

data structure. The data structures and operations are first formally defined. However, the language will be designed with the idea of embedding it in a procedural language for the purpose of control and other standard operations. In this paper, PL/I IBM (93) is the language chosen. Thus, the major criterion of the syntax design has been compatibility with PL/I. Any syntactical element not defined in this description will by default be assumed to be compatible with the PL/I definition.

We first illustrate some operations by referring to the physical representation corresponding to the abstract graph-structural entity. As an example, we consider the case in which each node represents a region of a picture or scene in two-dimensional space. Since the language allows any PL/I data structure to be associated with any element in the graph structure, physical data, such as a gray scale representation of a region, can be associated with a node. The procedural language also provides the capability of expressing mapping of attributes (which may represent structured data) of graph-structure elements during transformations.

After a brief listing of requirements and related work, the definition of SOL is given, followed by the syntax of SOL statements.

5.2 Language Requirements and Related Work

5.2.1 Requirements for a Graph-Structure Language

We here specify requirements for a graph-structure operation language for our picture processing application. These requirements will be listed in two parts: first, the elements of the structure, and secondly, the requirements of the elements necessitated by the dynamic nature of the processing.

Static Requirements

Basic Elements

Primitives (nodes)
Relations (branches) between pairs of nodes
Attributes and values for nodes
Attributes and values for branches

Sets of Elements

Arbitrary sets of nodes and branches
Graphs and subgraphs
Graph-levels (explicit substructures)
Pointers to elements

Dynamic Requirements

Add-Delete operations
Functions on attribute values
Pointer move operations
Structure replacement operations

5.2.2 Related Work

A number of graph manipulation languages and formalisms currently exist, but none of them has been found suitable for our purposes of simply and conveniently expressing operations on graph structures to satisfy the requirements listed in the preceding section. These graph processing languages and

systems include GASP of Chase (75), GRASPE of Pratt and Friedman (76), V-graphs of Earley (77), RSVP of Lieberman (78), the systems of Wolfberg (79), and Crespi-Reghizzi and Morpurgo (80).

GASP, the Graph Algorithm Software Package, is intended for basic operations on mathematical graphs (sets of nodes and edges) and is embedded in PL/I. We coded a region merging application quite simply, using GASP; it is well-suited for this type of problem, which does not require multiple labeled branches between nodes.

Hypergraphs in GRASPE and V-graphs permit a higher generality of branches. GRASPE includes basic operations to delete and create graph elements. However, SOL allows a greater complexity of branching and operation types.

The Relational Structure Vertex Processor, RSVP, employs a general data structure and operations on the structure. The basic element of the data structure is the atom. Atoms contain pointers to and from other atoms and a pointer to associated data. Operations are available to create or free cells or pointers, find atoms, allocate storage, store and retrieve parts of atoms and search by jumping between atoms. Since RSVP has only one type of pointer, information about different relation types must be forced into the type field of the atoms. This seems to be an unnatural restriction for our purposes. RSVP has features similar to many list processing languages and the basic operation seems to be lower-level than we desire in a graph processing language.

5.3 Definition of SOL

5.3.1 SOL Statements

The Structure Operation Language consists of statements called graph-structure statements which can be embedded in a PL/I program. SOL statements consist of declarations, associations, and operations. Declarations declare the basic structure elements. Associations declare and associate any PL/I variable with the basic structure elements. Operations are performed on the declared structures and are capable of dynamically creating new structures.

5.3.2 Basic Structure Elements

There are five basic structure elements which are designated by the attributes in a declaration. The elements are the pointer, set, node, branch, and graph.

A pointer points to or designates any other basic element. A pointer can designate a graph, branch, node, set, or pointer. Thus, a pointer allows indirect reference to any element.

A set designates a set of basic elements. It is a collection of pointers, each of which designates one element.

Nodes and branches are elements of graphs. A node designates a set of branches which are called adjacent branches of the node. A branch designates a pair of nodes. An oriented branch designates an ordered pair of nodes, the tail and the head of the branch. An unoriented branch designates an unordered pair of nodes.

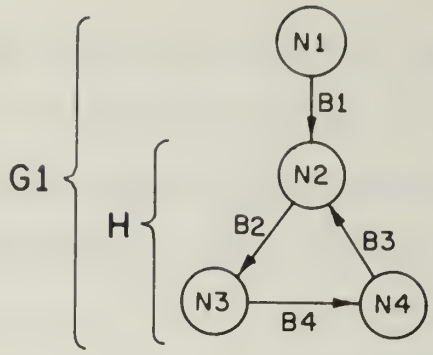
A graph is a set of nodes and branches with the property that if a branch belongs to a graph then its tail and head also belong to the graph. Graphs will be assumed to have oriented branches unless specifically declared otherwise.

5.3.3 Declarations

The DECLARE statement is used to declare the five basic structure elements. An element is declared by an identifier followed by an attribute which specifies the type of element. Declaring an element creates a new data element with the name given by the identifier. A level hierarchy is assumed within a declaration such that any graph element declared is assumed to belong to the last declared element at a lower numbered level. The level numbers are assigned so that GR < ND = BN. Also, graphs will be assumed to be subgraphs of the first preceding graph declared in the same declaration. Thus, nodes and branches will always belong to the last declared graph.

The attribute of a branch may be of the form BRANCH(N1,N2) where N1 and N2 are names for the tail and the head of the branch, respectively. This declaration will also declare the head and tail as nodes so that they need not be declared separately. The branch may also be declared without head and tail, which may be specified later, for example, by a LINK operation. Example declarations are given in Figure 19.

```
DECLARE
  G1 GRAPH,
  B1 BRANCH(N1,N2),
  H  GRAPH,
  B2 BRANCH(N2,N3),
  B3 BRANCH(N4,N2),
  B4 BRANCH(N3,N4);
```




```
DCL
  G2 GR,
  N(4) ND, B(2) BN,
  B(1) BN(N(2),N(1)), B(1) BN(N(4),N(3)),
  B(2) BN(N(1),N(3)), B(2) BN(N(2),N(4));
```

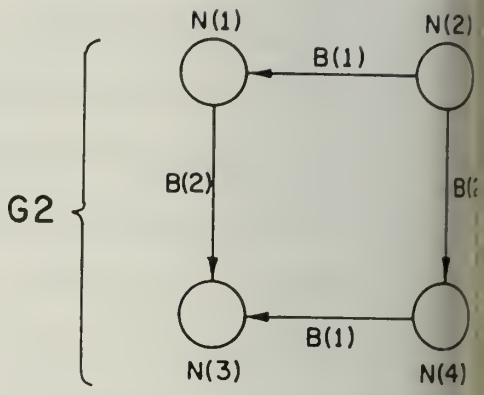


Figure 19. Example Declarations

5.3.4 Names and References to Elements

Names are not required for nodes and branches even though they are necessary in the DECLARE statement. Operations which dynamically modify a structure can result in the creation of a new unnamed element. Names need not be unique. For example, many branches can have the same name. Elements with non-unique names may be referenced uniquely, for instance by their position relative to other elements.

By a reference to an element, we mean a unique way of referring to the element. An element can be referenced by its name when the name is unique within the current scope or context. An element can always be referenced by a pointer,

since a pointer always points to only one element. Names for pointers and sets are always references, i.e. unique within the current scope, so a pointer or a set can always be referenced by a name.

Graphs, nodes, and branches, however, may have non-unique names. In this case, they may be referenced by a pointer, by a qualified name, or by other contextual means. A PL/I qualified name may be used for graphs, nodes, and branches using the level hierarchy assumed in the declare statement. A graph reference, for example,as used in a subgraph-operation, may explicitly list the elements of the subgraph. In this case, once the graph context is established, non-unique names may be used for references to nodes and branches.

5.3.5 Associations

An associate statement allows PL/I variables to be declared and associated with any or all basic structure elements of a given type. The free statement nullifies the association. Thus, an association is a function which can be defined on any or all elements of one type and which has a value of the type specified within the PL/I declaration.

The list of element-names specifies the elements to be associated with or freed from the variables which follow. If the element-name list is absent, all elements of the type specified are used. For example, given the example declaration of Figure 19, the associations:

```
BNASOC RELTYP BIN FIXED;
NDASOC (N(1) N(2) N(4))  XP BIN FIXED, YP BIN FIXED;
```

would associate a variable RELTYP of type fixed binary with all the branches of the graph G2 and would associate fixed binary variables XP and YP with the nodes N(1), N(2), and N(4). The statements BNFREE RELTYP; NDFREE XP, YP; would free the elements from the association.

The value of a variable associated with an element is accessed by using the variable (or function) name followed by an element reference enclosed in parentheses. In the preceding example, XP(N(2)) would refer to the value of the variable XP associated with the node N(2).

5.3.6 Data-Operation

Data-operations dynamically add or delete elements to or from previously declared graphs, subgraphs, and sets. A reference to the element being modified appears in parentheses following the operation name. The identifier and attributes which follow refer to new elements to be added or elements to be deleted.

Deleting an element from a set does not destroy the element but only deletes the reference to the element contained in the set. Similarly, deleting a node or branch from a subgraph will not destroy the node or branch, since it will still be a member of some graph. Deleting an element from a graph which is not a subgraph of another graph, however, will delete the

reference to the graph element and remove the element from existence.

For example, ADD(G1) N5 NODE, B5 BRANCH(N5,N2); will add one node and one branch to the graph G1 of the previous example.

DEL(G1) H; will delete the subgraph H from the graph G1 and remove all the nodes and branches of H from existence.

ADD(G2) S3 GRAPH, BA1 BRANCH(N(2), NN), BA2 BRANCH(NN,N(3)); will add a subgraph S3 with two new branches, BA1, BA2, and one new node, NN, to graph G2.

ADD(S) P; where S is a set and P is a pointer to some element, will add that element to the set S.

Deleting a node or a branch will also delete the node or branch from the sets which refer to it.

5.3.7 Loop-Control

The loop-control statement allows execution of the statements between the FOR statement and the END statement iteratively. FOR (i,v,s) specifies that each iteration will have a different value of the variable v chosen from the set s. The variable i specifies the number of iterations to be performed. ANY is equivalent to 1, and ALL is equivalent to the cardinality of the set s. Changing the set s within the loop

can change the number of iterations performed. For example, if an element x is deleted from s before x is used as the value of the loop-control variable v, then x will not be used.

5.3.8 Other Operations

Sets-set are binary operations on sets which return a set. The set returned is the union, intersection, difference, or symmetric difference of the given sets.

Sets-Boolean return a Boolean value corresponding to the truth or falsity of the statements "s_1 equals s_2," "s_1 is a subset of s_2," or "s_1 is a subgraph of s_2." The functions NODES and BRANCHES operate on a graph or set and return the set of nodes or the set of branches of the graph or set.

The INCBR, OUTBR, and ADJBR functions operate on a node and return the set of incoming, outgoing or adjacent branches of the node, respectively.

The HEAD and TAIL functions operate on a branch and return the head and tail of the branch respectively.

The CARD function operates on a set and returns an integer which is the cardinality of the set.

The NAME function returns the name of the element referenced. If a pointer name is used as the reference, then the name returned is the name of the element pointed to by the pointer.

The TYPE function returns the type of the element referenced. The type is 'GR,' 'ND,' 'BN,' 'PT,' or 'ST.'

5.3.9 Pointer-Operation

Pointer-operations change the value of a pointer. They are primarily useful for moving a pointer to adjacent nodes and branches of the current element pointed to, or for moving a pointer anywhere on a graph without having to use names.

The MOVE operation moves the pointer to an adjacent branch if it points to a node, or to an adjacent node if it points to a branch. If the branch (node) name is given, the pointer can only be moved to a branch (node) with the given name. If the named branch (node) does not exist as an adjacent branch (node), the pointer is not changed.

The OMOVE or "oriented move" operation is similar to MOVE except that moves will be made along branches only in the tail-to-head direction.

The JUMP operation moves the pointer to any node or branch on the graph. JUMP will move only from a node to a node or a branch to a branch so that the type of element pointed to is not changed. If no name is present, an arbitrarily picked node (branch) is used. If a name is present and no node (branch) with that name exists in the graph, then the pointer is not changed.

The MOVE2 operation is similar to two consecutive moves, so that a node pointer is moved to an adjacent node or a branch pointer is moved to an adjacent branch. MOVE2 is not equivalent to consecutive MOVE's since the operation will be performed completely or not at all. Thus, for example,

MOVE2 PTR1 RIGHT PLANE; will move a pointer named PTR1 along a branch named RIGHT to a node named PLANE only if both the branch and the node exist. However, the sequence MOVE PTR1 RIGHT; MOVE PTR1 PLANE; can result in only moving the pointer to a branch named RIGHT.

Assume a pointer named BUG points to a node. Then the operation:

MOVE BUG BRANCH2; moves BUG to an adjacent branch named BRANCH2.

MOVE BUG; moves BUG to an adjacent branch.

MOVE2 BUG; moves BUG to an adjacent node.

MOVE2 BUG RIGHT; moves BUG along a branch named RIGHT to an adjacent node.

5.3.10 Higher-Level Graph Operations

Node-operations and subgraph-operations both deal with subgraphs within a graph. Node-operations operate on a node and can partition a node into a subgraph or generate a subgraph sub-structure from a node. Subgraph-operations deal with a subgraph specified by a node set. The subgraph can be merged into one node, linked by "SUBPART" branches to one node, disconnected from all outside nodes, or be linked by a node chain. Figure 20 illustrates some of these operations.

The MERGE operation operates on all nodes of a specified subgraph and results in a new node. The new node may be named or may be pointed to by assigning the result of the MERGE to a node identifier or to a pointer, respectively. The

following actions occur when nodes XI are merged into a node X:

1. A new node, X, is created.
2. The node associations for X are the union of the associations for the set XI, and the value of each association is the same as the value for some arbitrary element of XI.
3. All branches between any element in XI and any element not in XI are changed to link between the same element not in XI and the new node X.
4. All nodes of the subgraph and all branches between nodes of the subgraph are deleted.

The PARSE operation is similar to the MERGE operation except that the merged subgraph is not deleted and the merged nodes are linked to the new node by "SUBPART" branches.

The PARTITION operation specifies a node and a partitioning of that node into nodes and branches. It is the inverse of the MERGE operation. The node specified is replaced by the subgraph and a procedure is necessary to specify the embedding (branches) between the subgraph and the graph.

The GENERATE operation is similar to the PARTITION operation except that the partitioned node is retained. The nodes of the subgraph specified are linked to the node specified by "SUBPART" branches.

The DISCONNECT operation disconnects all branches between any node outside and any node inside the set of nodes specified. If a branch name is specified, only branches with the given name are deleted. SINK and SOURCE operate like

DISCONNECT except that SINK does not disconnect branches directed into the node set, and SOURCE does not disconnect branches directed out of the node set.

The LINK operation links all nodes into an arbitrarily ordered chain by branches having the specified name. If no branch name is specified, unnamed branches are used.

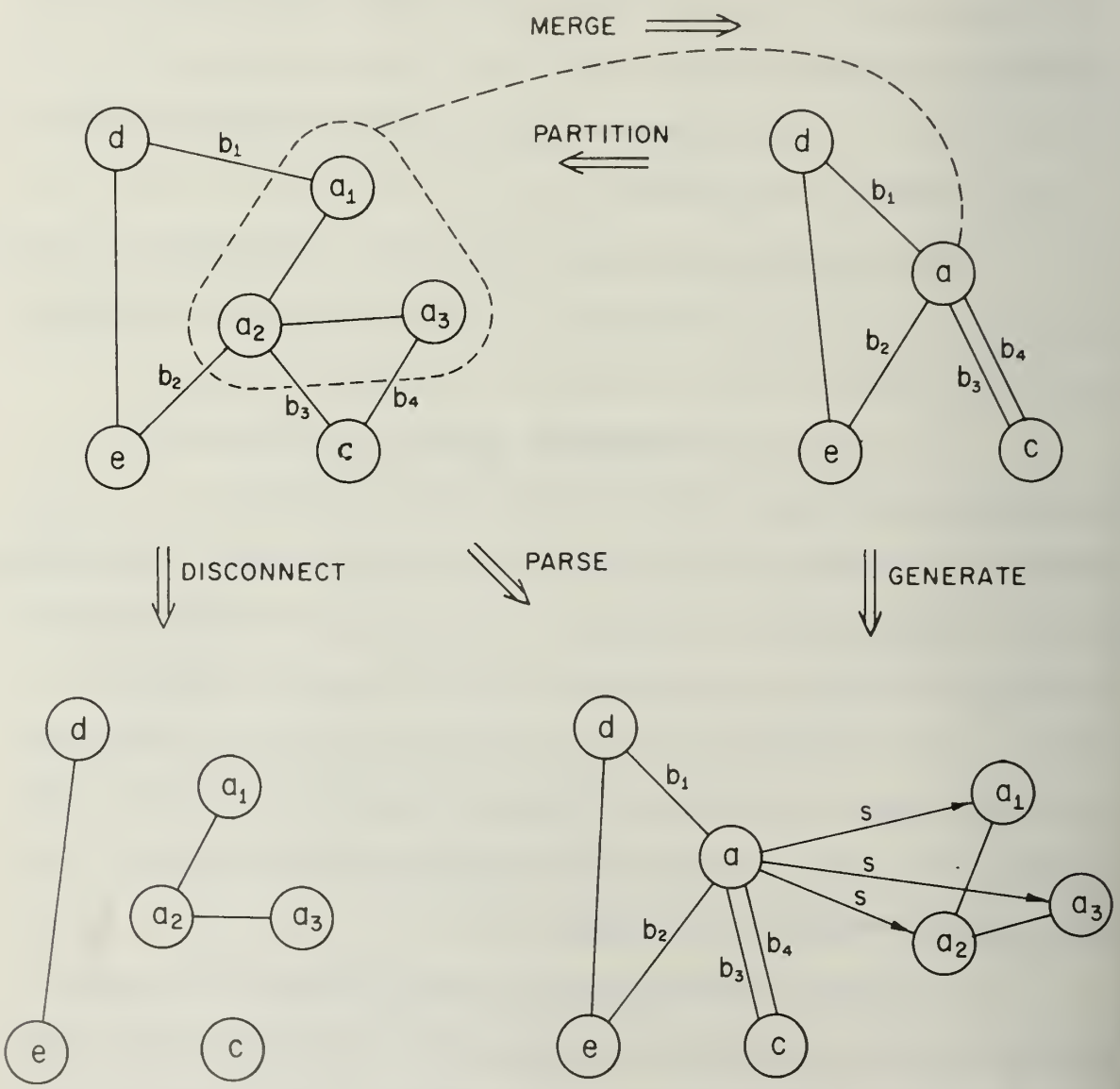


Figure 20 . SOL Subgraph Operations

5.4 SOL Syntax

The syntax of SOL is given below in PL/I syntax notation. Any nonterminal beginning with "pl1" is the same as the corresponding PL/I construct and is not defined here.

```
sol-program ::= statement  [ ; statement ] ...
statement ::= declaration | associate | free | operation
        |if-statement | pl1-statement
declaration ::=  { DCL | DECLARE }  element-declaration
        [, element-declaration ] ...
element-declaration ::= identifier [ (pl1-bound [, pl1-bound]...)]
        element-attribute
element-attribute ::= { GR | GRAPH }  [UNORIENTED] | {ND | NODE}
      | {BN | BRANCH } [( node-reference, node-reference )]
      | {PT | POINTER}  |  {ST | SET}
IF STATEMENT ::= IF pl1-expression THEN statement
        [; ELSE statement]
associate ::= { GRASOC | NDASOC | BNASOC | PTASOC | STASOC }
        [( qualified-name...) ]  pl1-declaration-tail
free ::= {GRFREE | NDFREE | BNFREE | PTFREE | STFREE }
        [( qualified-name...)]  identifier [, identifier]...
```

operation ::= loop-control | data-operation | sets-set
| sets-Boolean | graph-set | node-set | branch-node
| set-integer | element-string | pointer-operation
| node-operation | subgraph-operation

data-operation ::= {ADD | DEL} (graph-reference | set-
reference) {element-declaration | element-reference}
[, element-declaration | element-reference]...

loop-control ::= FOR ({ANY | ALL | integer}, identifier,
set-reference) | END

sets-set ::= sets-set-mnemonic (set-reference, set-reference)

sets-set-mnemonic ::= UNION | INTER | DIFF | SYMDIFF

sets-Boolean ::= sets-Boolean-mnemonic (set-reference,
set-reference)

sets-Boolean-mnemonic ::= EQUALS | SUBSET | SUBGRAPH

graph-set ::= graph-set-mnemonic (graph-reference | set-reference)

graph-set-mnemonic ::= NODES | BRANCHES

node-set ::= node-set-mnemonic (node-reference)

node-set-mnemonic ::= INCBR | OUTBR | ADJBR

branch-node ::= branch-node-mnemonic (branch-reference)

branch-node-mnemonic ::= HEAD | TAIL

set-integer ::= set-integer-mnemonic (set-reference)

```
set-integer-mnemonic ::= CARD

element-string ::= element-string-mnemonic  (element-reference)

element-string-mnemonic ::= NAME | TYPE

element-reference ::= graph-reference | set-reference
        |node-reference | branch-reference | pointer-reference

graph-reference ::= qualified-name | pointer-reference

node-reference ::= branch-node | qualified-name | pointer-reference

branch-reference ::= qualified-name | pointer-reference

set-reference ::= sets-set | graph-set | node-set | name
        |pointer-reference

pointer-reference ::= name

qualified-name ::= name [. name]...

name ::= identifier   [(pl1-subscript [, pl1-subscript] ...)]

pointer-operation ::= {JUMP | MOVE | OMOVE} pointer-reference
          [branch-reference | node-reference]|{MOVE2 | OMOVE2}
           pointer-reference [branch-reference[node-reference]
         |node-reference [ branch-reference] ]

node-operation ::= {PARTITION | GENERATE} node-reference
        graph-reference  pl1-procedure-call

subgraph-operation ::= { MERGE | PARSE | DISCONNECT | SINK
       |SOURCE | LINK } {node-reference | set-reference}...
        [branch-reference]
```

5.5 Example Program Segments

Find the set S of all nodes in a given graph G which are linked to one node named N by incoming branches named BA:

```
DECLARE     B PT, S ST;
            FOR( ALL , B , INCBR(N) ) ;
                 IF NAME(B) = 'BA' THEN
                 ADD(S) TAIL(B) ;
            END ;
```

General pairwise merge of all linked nodes in a graph G under some condition D(B) until no further merges are possible:

```
RELOOP :    N = CARD( NODES( G ) ) ;
            FOR( ALL , B , BRANCHES( G ) ) ;
                 IF D(B) THEN
                 MERGE TAIL( B )   HEAD( B ) ;
            END ;
            IF CARD( NODES( G ) )¬ = N THEN
            GO TO RELOOP ;
```

6. SUMMARY AND CONCLUSIONS

6.1 Discussion of Results

The main results of this work are: the definition of a graph-structure transformation model, the development of formal types of embedding properties of a relation under a transformation, and the definition of a graph-structure operation language.

The graph-structure model provides a framework in which to express multi-relational scene segmentation and structural inference techniques. The graph-structure representation and transformations are defined as having properties to facilitate their application to the parsing of pictures.

Simple embedding concepts of relations have been considerably expanded so that relations among submembers of composite structures can infer relations between the composite structures. Picture processing applications of embedding types have been demonstrated.

A structure operation language, SOL, has been defined to provide a means of experimenting with heuristic structure transformation procedures; in particular, SOL is intended to provide an implementation of the graph-structure model.

6.2 Suggestions for Further Research

The study of the application of graph transformations to picture parsing involves the following areas, all open for investigation:

Choice of domain for the transformations

Expression of the semantics of rules

Properties of relations

Optimality of a transformation or of a parse

Heuristics to choose where to apply a transformation should include attribute classification techniques. Attributes will undoubtedly express the semantics of the structural rules which we have considered. The study of how attributes act under a transformation is indicated.

The formal logical properties of binary relations (such as defined in McCormick and Schwebel (25)) could be employed in a methodology to infer or delete relations in a structure. The development of properties of relations to infer transformations is needed. Abstract embedding type properties could infer the relative stength of a relation with respect to a particular transformation. Negative embedding types, such as "NOT" which we considered, should be extended to composites. Particular picture-processing relations could be categorized with respect to the formal properties studied, and then applied.

Criteria to determine optimality of a single transformation, or of a parse composed of a series of transformations are needed. Such criteria should include: the shortest

series and the simplest transformations among those which obtain valid descriptions.

The feasibility of structure transformation heuristic techniques for processing abstract graphs of simple scenes has been demonstrated by some simple programs. Further research should develop explicit algorithms for picture processing and for other related classes of structural inference problems:

Common structure between graphs

Common structure between sets of graphs

Structural analogies

Finally, the implementation of SOL, which is already underway in this Department, could be developed appropriately for its use at a graphics console for interactive experimentation with the algorithms mentioned above.

LIST OF REFERENCES

(1) A. Rosenfeld, Picture Processing by Computer, Academic Press, New York, 1969.

(2) G. Nagy, "State of the art in pattern recognition," Proceedings of the IEEE, vol. 56, 1968, pp. 836-862.

(3) M. D. Levine, "Feature extraction: a survey," Proceedings of the IEEE, vol. 57, no. 8, August, 1969, pp. 1391-1407.

(4) R. Paul, G. Falk, and J. A. Feldman, "The computer representation of simply described scenes," in M. Faiman and J. Nievergelt (Eds.), Pertinent Concepts in Computer Graphics, U. of I. Press, 1969, pp. 87-103.

(5) D. Hsiao and F. Harary, "A formal system for information retrieval from files," Communications of the ACM, vol. 13, no. 2, February, 1970, pp. 67-73.

(6) R. H. Anderson, "Syntax-directed recognition of handprinted two-dimensional mathematics," PhD. Thesis, Harvard University, January, 1968.

(7) T. G. Evans, "A grammar controlled pattern analyzer," Information Processing 68, Proceedings of the IFIP Congress, Booklet H, 1968, pp. 152-157.

(8) J.Feder, "Languages of encoded line patterns," Information and Control, vol. 13, 1968, pp. 230-244.

(9) J. Feder, "Linguistic specification and analysis of classes of line patterns," School of Engineering, EE Department, New York University, April, 1969.

(10) A. C. Shaw, "Parsing of graph-representable pictures," Journal of the ACM, vol. 17, no. 3, July, 1970, pp.453-481.

(11) W. F. Miller and A. C. Shaw, "Linguistic methods in picture processing - a survey," AFIPS Fall Joint Computer Conference, 1968, pp. 279-290.

(12) J. Feder, "The linguistic approach to pattern analysis: a survey," Report no. 400-133, School of Engineering, EE Department, New York University, February, 1966.

(13) P. H. Swain and K. S. Fu, "Nonparametric and linguistic approaches to pattern recognition," TR-EE 70-20, School of EE, Purdue University, June, 1970.

(14) R. Narasimhan, "Picture languages," Conference on Picture Language Machines, Australian National University, Canberra, Australia, February, 1969.

(15) R. Narasimhan, "On the description, generation and recognition of classes of pictures," Summer School on Automatic Interpretation and Classification of Images, NATO Advanced Study Institute, Pisa, Italy, August 26-September 7, 1968.

(16) R. A. Kirsch, "Computer interpretation of English text and picture patterns," EC-13, no. 4, IEEE Trans. on Electronic Computers, August, 1964, pp. 363-376.

(17) S. Chang, "A method for the structural analysis of two-dimensional mathematical expressions," no. RC 2655, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, October, 1969.

(18) S. Chang, "The analysis of two-dimensional patterns using picture processing grammars," Second ACM Symposium on Theory of Computing, May, 1970, pp. 206-216.

(19) P. J. Knoke and R. C. Wiley, "A linguistic approach to mechanical pattern recognition," Proceedings of the IEEE Computer Conference,September, 1967, pp. 142-144.

(20) D. F. Londe and R. F. Simmons, "NAMER: a pattern recognition system for generating sentences about relations between line drawings," Proceedings of the ACM Twentieth National Conference, 1965, pp. 162-175.

(21) A. Inselberg and R. Kline, "SAP: a model for the syntactic analysis of pictures," Technical Report no. 9, Computer Systems Laboratory, Washington University, June, 1968.

(22) A. Inselberg, "An approach to the syntax-directed analysis of graphic data,"Technical memo 52, Computer Systems Laboratory, Washington University, October, 1967.

(23) M. B. Clowes, "Pictorial relationships- a syntactic approach," in B. Meltzer and D. Michie (Eds.), Machine Intelligence 4, Edinburgh University Press, 1969, pp. 361-383.

(24) H. G. Barrow and R. J. Popplestone, "Relational descriptions in picture processing," in B. Meltzer and D. Michie (Eds.), Machine Intelligence 6, Edinburgh University Press, 1971, pp. 325-375.

(25) B. H. McCormick and J. C. Schwebel, "Consistent formal properties of binary relations," DCS File no. 762, University of Illinois at Urbana-Champaign, July, 1968.

(26) B. Raphael, "A computer program which understands,"AFIPS, Proceedings of the Fall Joint Computer Conference, 1964, pp. 577-589.

(27) D. L. Childs, "Feasibility of a set-theoretic data structure," Technical Report no. 6, CONCOMP, University of Michigan, August, 1968.

(28) D. L. Childs, "Description of a set-theoretic data structure," Technical Report no. 3, CONCOMP, University of Michigan, August, 1968.

(29) R. W. Elliot, "A model for a fact retrieval system," no. TNN-42, University of Texas, Computation Center, May, 1965.

(30) C. M. Eastman, "Explorations of the cognitive processes in design, "Computer Science Department, Carnegie-Mellon University, February, 1968.

(31) C. M. Eastman, "Representations for space planning," Communications of the ACM, vol. 13, no. 4, April, 1970, pp. 242-250.

(32) D. A. Savitt, H. H. Love, and R. E. Troop, "ASP: a new concept in language and machine organization," Proceedings of the the Spring Joint Computer Conference, 1967, pp. 87-102.

(33) G. U. Montanari, "Networks of constraints: fundamental properties and applications to picture processing," Department of Computer Science, Carnegie-Mellon University, January, 1971.

(34) R. Williams, "A survey and an annotated bibliography of data structures for computer graphics systems," TR 403-6, EE Department, New York University, September, 1969.

(35) J. C. Gray, "Compound data structures for computer aided design - a survey," Proceedings of the Twenty-second National ACM Conference, ACM Publication P-67, 1967, pp. 355-365.

(36) T. G. Evans, "Descriptive pattern-analysis techniques: potentialities and problems," in W. Satosi (Ed.) Methodologies of Pattern Recognition, Academic Press, 1969, pp. 147-157.

(37) R. O. Duda and P. E. Hart, "A survey of pattern classification and scene analysis," Stanford Research Institute, January, 1971.

(38) L. E. Lipkin, W. C. Watt, and R. A. Kirsch, "The analysis, synthesis and description of biological images," Annals of the New York Academy of Sciences, vol. 128, article 3, 1966, pp. 984-1012.

(39) A. Guzman, "Computer recognition of three-dimensional objects in a visual scene,"MAC TR-59, Project MAC, MIT, Decmeber, 1968.

(40) A. Guzman, "Decomposition of a visual scene into three-dimensional bodies," AFIPS Fall Joint Computer Conference, vol. 33, 1968, pp. 291-304.

(41) G. Falk, "Computer interpretation of imperfect line data as a three-dimensional scene," Report no. CS180, Computer Science Department, Stanford University, August, 1970.

(42) D. A. Huffman, "Logical analysis of pictures of polyhedra," Tech. Note no. 6, Artificial Intelligence Group, Stanford Research Institute, May, 1969.

(43) D. A. Huffman, "Impossible objects as nonsense sentences," in B. Meltzer and D. Michie (Eds.), Machine Intelligence 6, Edinburg University Press, 1971.

(44) C. R. Brice and C. L. Fennema, "Scene analysis using regions," Artificial Intelligence, vol. 1, 1970, pp. 205-226.

(45) T. Pavlidis, "Computer recognition of figures through decomposition," Information and Control,vol. 12, 1968, pp. 526-537.

(46) T. Pavlidis, "Computer analysis of figures into primary convex subsets," Record of the IEEE SSC Conference, Cat. no. 68C23-SSC, 1968, pp. 55-60.

(47) R. S. Michalski and B. H. McCormick,"Interval generalization of switching theory," Report no. 442, Department of Computer Science, University of Illinois at Urbana-Champaign, May, 1971.

(48) R. S. Michalski, "A variable-valued logic system as applied to picture description and recognition," to appear in Proceedings of the IFIP Graphic Languages Conference, North Holland Publishing Co., 1972

(49) P. Raulefs, Thesis in preparation, Department of Computer Science, University of Illinois at Urbana-Champaign.

(50) A. Guzman, "Some aspects of pattern recognition by computer," MAC TR-37, Project MAC, MIT,February, 1967.

(51) A. Guzman, "Analysis of curved line drawings using context and global information," in B. Meltzer and D. Michie (Eds.), Machine Intelligence 6, 1971, pp.325-375.

(52) F. P. Preparata and S. R. Ray, "An approach to artificial nonsymbolic cognition," Report no. R478, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, July, 1970.

(53) G. Salton and E. H. Sussenguth, "Some flexible information retrieval systems using structure matching procdures," AFIPS, Proceedings of the Spring Joint Computer Conference, 1964, pp. 587-597.

(54) H. Freeman and L. Gardner, "Apictorial jigsaw puzzles: the computer solution to a problem in pattern recognition," IEEE Transactions on Computers, vol. EC-13, no. 2, 1964, pp. 118-127.

(55) D. W. Matula, "Cluster analysis via graph theoretic techniques," in R. C. Mullin, K. B. Reid, and D. P. Roselle (Eds.), Proceedings of the Louisiana Conference on Graph Theory Combinatorics and Computing, 1970, pp. 199-212.

(56) C. T. Zahn, "Graph-theoretical methods for detecting and describing gestalt clusters," IEEE Transactions on Computers, vol. C-20, no. 1, January, 1971, pp. 68-86.

(57) J. L. Pflantz, "Convexity in graphs," TR-68-74, Computer Science Center, University of Maryland, July, 1968.

(58) J. L. Pflantz and A. Rosenfeld, "Web grammars," Proceedings International Joint Conference on Artificial Intelligence, May, 1969, pp. 609-619.

(59) G. U. Montanari, "Separable graphs, planar graphs, and web grammars," Information and Control, vol. 16, no. 3, May, 1970, pp. 243-267.

(60) T.Pavlidis, "Linear and context-free graph grammars," Journal of the ACM, vol. 19, no. 1, January, 1972.

(61) R. A. Kirsch, "Computer determination of the constituent structure of biological images, part 1," NBS Report no. 10173, National Bureau of Standards, US Department of Commerce, December, 1969.

(62) T. G. Evans, "A heuristic program to solve geometric-analogy problems," AFIPS, Proceedings of the Spring Joint Computer Conference, 1964, pp. 327-328.

(63) T. G. Evans, "Grammatical inference techniques in pattern analysis," Proceedings of the Third COINS Symposium, December, 1969.

(64) P. H. Winston, "Learning structural descriptions from examples," MAC-TR-76, Project MAC, MIT, September, 1970.

(65) D. C. Cooper, "Some transformations and standard forms of graphs with applications to computer programs," in Dale and D. Michie (Eds.), Machine Intelligence 2, Edinburgh University Press, 1968, pp. 21-32

(66) S. K. Basu, "Transformations of program schemes to standard forms," Department of Computer Science, Carnegie-Mellon University, February, 1968.

(67) B. H. McCormick, "Syntax-directed recognition of pictures: a new theoretical model," presented at the Computer Graphics Conference, University of Illinois at Urbana-Champaign, November, 1967.

(68) J. C. Schwebel, "Use of graph transformations to characterize an image: an illustrative example," File no. 770, Department of Computer Science, University of Illinois at Urbana-Champaign, July, 1968.

(69) J. C. Schwebel, "Graph transformations for composite formation," DCS Report no. 368, University of Illinois at Urbana-Champaign, Department of Computer Science, December, 1969.

(70) J. C. Schwebel and B. H. McCormick, "Consistent Properties of composite formation under a binary relation," Information Sciences, vol. 2, no. 2, April, 1970, pp. 179-209.

(71) B. Herzog, "Lectures on computer graphics," Computer and Program Organization- Fundamentals, University of Michigan Engineering Summer Conferences, June, 1967.

(72) H. E. Kulsrud, "A general purpose graphic language," Communications of the ACM, vol. 11, no. 4, April, 1968, pp. 247-254.

(73) R. Williams, "A systematic method for the creation of data structures in computer graphics applications,"Tech. Report 403-19, EE Department, New York University, April, 1971.

(74) J. C. Schwebel, "Towards the specification of a new image processing language," DCS File no. 788, University of Illinois at Urbana-Champaign, February, 1969.

(75) S. M. Chase, "Analysis of algorithms for finding all spanning trees of a graph," DCS Report no. 401, University of Illinois at Urbana-Champaign, 1970.

(76) T. W. Pratt and D. P. Friedman, "A language extension to graph processing and its formal semantics," Communications of the ACM, vol. 14, no. 7, July, 1971, pp. 460-467.

(77) J. Earley, "Toward an understanding of data structures," Communications of the ACM, vol. 14, p. 617, 1971.

(78) R. N. Lieberman, "RSVP relational structure vertex processor," Tech. Report no. 69-87, Computer Science Center, University of Maryland, March, 1969.

(79) M. S. Wolfberg, "An interactive graph theory system," Report no. 69-25, Moore School of EE, University of Pennsylvania, June, 1969.

(80) S. Crespi-Reghizzi and R, Morpurgo, "A language for treating graphs," Communications of the ACM, vol. 13, no. 5, May, 1970, pp. 319-323.

(81) D. E. Knuth, "Semantics of context free languages," Mathematical Systems Theory, vol. 2, no. 2, 1968, pp. 127-145.

(82) D. E. Knuth, "Examples of formal semantics," Report no. CS 169, Computer Science Department, Stanford University, July, 1970.

(83) D. L. Milgram and A. Rosenfeld, "A note on grammars with coordinates," Tech. Report no. 70-140, Computer Science Center, University of Maryland, September, 1970.

(84) G. W. Ernst and A. Newell, "Generality and G. P. S," Carnegie Institute of Techology, Pittsburgh, January,1967.

(85) L. Siklossy, "Generalized means-ends analysis and artificial intelligence," Information Sciences, vol. 3, 1971, pp. 149-158.

(86) A. D. C. Holden and D. L. Johnson, "The use of embedded patterns and canonical forms in a self-improving problem solver," Proceedings of the ACM National Meeting, 1967, pp. 211-219.

(87) J. R. Slagle and P. Bursky, "Experiments with a multi-purpose, theorem proving heuristic program," Journal of the ACM, vol. 15,no.1, January, 1968, pp. 85-99.

(88) P. H. Winston, "A heuristic program that constructs decision trees," AI Memo 173, Project MAC, MIT, March,1969.

(89) G. A. Gorry, "A system for computer-aided diagnosis," MAC-TR-44,(Thesis), Project MAC, MIT, September, 1967.

(90) G. A. Gorry and G. O. Barnett, "Experience with a model of sequential diagnosis," Computers and Biomedical Research, vol.1, 1968, pp. 490-507.

(91) G. Birkhoff, Lattice Theory, American Mathematical Society, Providence, Rhode Island, 1963.

(92) G. Szasz, Introduction to Lattice Theory, Academic Press, New York, 1963.

(93) IBM, PL/I(F) Reference Manual, Systems Reference Library, Order no. GC28-8201-3, 1970.

VITA

John Charles Schwebel was born in St. Paul, Minnesota, on June 30, 1942. He graduated from the University of Minnesota Institute of Technology with a Bachelor of Mathematics degree in 1964.

From 1964-1971, he was a research assistant in the Department of Computer Science at the University of Illinois, working on the Illiac III Computer Project. He received his Master of Science degree in Mathematics in 1966.

Since September of 1971, he has been with the Data Processing Division of Univac at Roseville, Minnesota.

Mr. Schwebel is the co-author with Bruce H. McCormick of a paper entitled "Consistent Properties of Composite Formation Under a Binary Relation," which evolved from his thesis research.

Mr. Schwebel is a member of the Association for Computing Machinery and the Society of the Sigma Xi.

Form AEC-427
(6/68)
AECM 3201

U.S. ATOMIC ENERGY COMMISSION
UNIVERSITY-TYPE CONTRACTOR'S RECOMMENDATION FOR DISPOSITION OF SCIENTIFIC AND TECHNICAL DOCUMENT

(See Instructions on Reverse Side)

1. AEC REPORT NO.
COO-2118-0031
UIUCDCS-R-72-514

2. TITLE
A GRAPH-STRUCTURE TRANSFORMATION MODEL FOR PICTURE PARSING

3. TYPE OF DOCUMENT (Check one):

- [X] a. Scientific and technical report
- [ ] b. Conference paper not to be published in a journal:
  - Title of conference _____
  - Date of conference _____
  - Exact location of conference _____
  - Sponsoring organization _____
- [ ] c. Other (Specify) _____

4. RECOMMENDED ANNOUNCEMENT AND DISTRIBUTION (Check one):

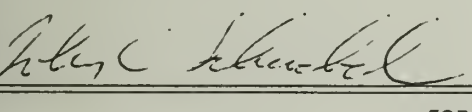
- [X] a. AEC's normal announcement and distribution procedures may be followed.
- [ ] b. Make available only within AEC and to AEC contractors and other U.S. Government agencies and their contractors.
- [ ] c. Make no announcement or distrubution.

5. REASON FOR RECOMMENDED RESTRICTIONS:

6. SUBMITTED BY: NAME AND POSITION (Please print or type)

John C. Schwebel
Ph.D. Thesis Candidate

Organization
Department of Computer Science
University of Illinois
Urbana, Illinois 61801

Signature 

Date May 8, 1972

FOR AEC USE ONLY

7. AEC CONTRACT ADMINISTRATOR'S COMMENTS, IF ANY, ON ABOVE ANNOUNCEMENT AND DISTRIBUTION RECOMMENDATION:

PATENT CLEARANCE:

- [ ] a. AEC patent clearance has been granted by responsible AEC patent group.
- [ ] b. Report has been sent to responsible AEC patent group for clearance.
- [ ] c. Patent clearance not required.

BLIOGRAPHIC DATA EET	1. Report No. COO-2118-0031 UIUCDCS-R-72-514	2.	3. Recipient's Accession No.

Title and Subtitle

A GRAPH-STRUCTURE TRANSFORMATION MODEL FOR PICTURE PARSING

5. Report Date: May, 1972

6.

Author(s)

John C. Schwebel

8. Performing Organization Rept. No.

Performing Organization Name and Address

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

10. Project/Task/Work Unit No.: Illiac III

11. Contract/Grant No.: AT(11-1)-2118

Sponsoring Organization Name and Address

U. S. Atomic Energy Commission

13. Type of Report & Period Covered: Ph.D. Thesis

14.

Supplementary Notes

Abstracts: This paper develops a model which defines graph structures as representations and graph-structure transformations as operations. The intended application is to picture analysis, and in particular to the inference of syntactic and semantic structure from instances of objects and scenes. Representative graph transformations are defined. Necessary properties of transformations are developed, and reversibility and information loss are defined. In order to see how graph branches are inferred, simple and composite embedding types of binary relations under a transformation are studied. Lattice operational properties are defined to characterize relations between lattice elements by the way they act when the lattice is transformed. The goal of using graph transformations to obtain a simplified description is discussed. A computer language, SOL, is defined to implement graph structures and graph-structure transformations.

Key Words and Document Analysis. 17a. Descriptors

. Identifiers Open-Ended Terms

COSATI Field/Group

Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 124
Release Unlimited	20. Security Class (This Page UNCLASSIFIED	22. Price

BOUND BY
HNM

UNIVERSITY OF ILLINOIS-URBANA
510.84 IL6R no. C002 no.511-516(1972
RASER: random to serial converter.
3 0112 088400368